TUGAS AKHIR – KI141502

RANCANG BANGUN WEB SERVICE UNTUK IMPLEMENTASI ATURAN MAIN DAN MANAJEMEN TRANSAKSI PADA PERMAINAN SOSIAL PERANGKAT BERGERAK CARD WARLOCK SAGA

MUHAMMAD AGIL MAHBUBY
NRP 5110 100 191

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Abdul Munif, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

FINAL PROJECT – KI141502

WEB SERVICE DESIGN FOR GAME RULES AND TRANSACTION MANAGEMENT IMPLEMENTATION ON MOBILE SOCIAL GAME CARD WARLOCK SAGA

MUHAMMAD AGIL MAHBUBY
NRP 5110 100 191

Advisor
Imam Kuswardayan, S.Kom., M.T.
Abdul Munif, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2015

KATA PENGANTAR

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan tugas akhir yang berjudul :

"Rancang Bangun Web Service untuk Implementasi Aturan Main dan Manajemen Transaksi pada Permainan Sosial Perangkat Bergerak Card Warlock Saga"

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Bapak Musthofa Zahron, Ibu Nur Abadiyah, Ibu Nur Maimunah, Mbak Vicky, Mas Ibrahim, serta segenap keluarga yang senantiasa memberikan doa dan dukungan untuk menyelesaikan tugas akhir ini;
2. Bapak Imam Kuswardayan dan Bapak Abdul Munif selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberikan ilmu dan biimbingan selama proses pengerjaan tugas akhir ini;
3. serta teman-teman seperjuangan: Gegi, Fauzi, Firman, Misbah, Ichank, Zendra, Ardian, Rizki, dan teman-teman lainnya yang tidak dapat disebutkan satu per satu.

Penulis berharap semoga apa yang ada di buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan serta dapat memberikan inspirasi bagi pembaca.

Surabaya, Januari 2015

Muhammad Agil Mahbuby

LEMBAR PENGESAHAN

Rancang Bangun *Web Service* untuk Implementasi Aturan Main dan Manajemen Transaksi pada Permainan Sosial Perangkat Bergerak *Card Warlock Saga*

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Interaksi, Grafik dan Seni
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
MUHAMMAD AGIL MAHBUBY
NRP : 5110 100 191

Disetujui oleh Dosen Pembimbing tugas akhir :

IMAM KUSWARDAYAN, S.Kom., M.T. .....
NIP: 197612152003121001 (pembimbing 1)

ABDUL MUNIF, S.Kom., M.Sc. .....
NIP: 051100114 (pembimbing 2)

SURABAYA
JANUARI 2015

Rancang Bangun *Web Service* untuk Implementasi Aturan Main dan Manajemen Transaksi pada Permainan Sosial Perangkat Bergerak *Card Warlock Saga*

Nama Mahasiswa	: Muhammad Agil Mahbuby
NRP	: 5110100191
Jurusan	: Teknik Informatika FTIf-ITS
Dosen Pembimbing 1	: Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing 2	: Abdul Munif, S.Kom., M.Sc.

ABSTRAK

Web service digunakan dalam sebuah aplikasi untuk mengatur jalannya pertukaran data antara aplikasi dengan data yang ada pada server. Aplikasi dapat secara online mengakses fungsi-fungsi yang ada pada web service dan merequest data yang diminta untuk dikembalikan kepada aplikasi tersebut. Web service dapat dibangun dari berbagai macam bahasa pemrograman, salah satunya adalah bahasa pemrograman PHP. Dalam proses pengiriman datanya, web service akan menserialisasikan data tersebut terlebih dahulu. Bentuk data yang umum digunakan adalah XML dan JSON.

Card Warlock Saga menggunakan layanan web service dalam bahasa pemrograman PHP untuk mengatur pertukaran data dari klien ke database server. Aplikasi klien pada Unity dapat mengakses fungsi-fungsi yang telah disediakan oleh web service, di mana kemudian fungsi pada web service tersebut akan melanjutkan pemrosesan data terhadap database yang ada pada server. Respon dari database dan web service ini nantinya akan dikembalikan kepada aplikasi klien yang ada pada Unity dalam bentuk data yang telah diserialisasikan dalam bentuk XML.

***Kata kunci**: Game, Social Game, Web Service.*

Web Service Design and Implementation of Game Rules and Transaction Management on Social Game Card Warlock Saga

Student Name	: Muhammad Agil Mahbuby
Student ID	: 5110100191
Major	: Teknik Informatika FTIf-ITS
Advisor 1	: Imam Kuswardayan, S.Kom., M.T.
Advisor 2	: Abdul Munif, S.Kom., M.Sc.

ABSTRACT

Web service is used by an application to control the process of the data flow between the application and the server. The application is able to access the functions on the web service online, in order to request the data needed, then it is sent back by the web service to the application to be used. Web service can be developed in several programming language, including PHP. In processing the data, web service will serialize the data in order to be sent and received by the client. The general serialization of data are XML and JSON.

Card Warlock Saga uses a PHP web service to control the data flow process betweeen the client and database server. The client application in Unity is able to access the functions served by web service, and it will continue the data process to the database. The response from both database and web service is then transported back to Unity client in XML serialized data form.

Keyword**: **Game, Social Game, Web Service.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1. Latar Belakang

Permainan video telah berkembang cepat dan modern, di mana dahulu permainan video hanya dimainkan oleh beberapa kalangan saja, sekarang permainan video hampir dimainkan oleh seluruh kalangan. Permainan video pada saat ini juga menyertakan teknologi *broadband* yang berguna untuk mengakses internet.

Teknologi informasi yang berkembang dengan sangat cepat memungkinkan permainan video untuk dimainkan di perangkat bergerak seperti telepon genggam pintar. Perkembangan sistem operasi Android sebagai salah satu sistem operasi telepon genggam yang populer menyebabkan banyaknya pengguna telepon genggam berbasis sistem operasi tersebut.

Saat ini mulai bermunculan banyak situs-situs penyedia layanan jejaring sosial seperti Facebook dan Twitter. Dengan jumlah pengguna yang cukup banyak, dimungkinkan untuk menciptakan permainan video yang terintegrasi dengan jejaring sosial yang dinamakan dengan permainan sosial. Tentunya aplikasi nantinya akan diintegrasikan dengan jejaring sosial yang telah dipilih. Integrasi dalam hal ini berarti penyinkronan data antara data yang ada pada *client* dengan data yang ada pada *server*. Maka dari itulah diperlukan sebuah *web service* untuk mengatur pertukaran data antara *server* dengan *client*.

1.2. Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah untuk membuat *web service* yang dapat diintegrasikan dengan modul-modul lain pada permainan sosial *Card Warlock Saga*.

1.3. Rumusan Permasalahan

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut.

1. Bagaimana membuat aplikasi *web service* yang dapat berjalan pada permainan sosial *Card Warlock Saga*?
2. Bagaimana merancang dan mengimplementasikan aturan main dan manajemen transaksi pada permainan sosial *Card Warlock Saga* dalam bentuk *stored procedure*?
3. Bagaimana membuat arsitektur *web service* dan fungsi-fungsinya agar dapat diakses oleh modul-modul lain pada permainan sosial *Card Warlock Saga*?
4. Bagaimana membuat manajemen transaksi basis data pada permainan sosial *Card Warlock Saga* agar dapat diakses secara *massive*?
5. Bagaimana mengimplementasikan aturan main dan manajemen transaksi agar dapat diuji pada *layer database* sebelum diintegrasikan pada modul-modul lain?

1.4. Batasan Permasalahan

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, di antaranya adalah sebagai berikut.

1. Bahasa pemrograman yang digunakan untuk membangun *web service* ini adalah PHP dengan menggunakan *framework* CodeIgniter.

2. *Web service* yang dibangun menggunakan MySQL sebagai sistem manajemen basis data.
3. *Web service* yang dibangun hanya digunakan untuk permainan *Card Warlock Saga*.

1.5. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan tugas akhir ini yaitu sebagai berikut.

1. Studi literatur

Pada tahap ini dilakukan pengumpulan informasi dan referensi yang diperlukan dalam proses perancangan dan implementasi aplikasi yang akan dibangun, yaitu sebagai berikut.

a. Konsep dari jenis-jenis arsitektur *web service*.
b. Konsep serialisasi data dalam bentuk XML.
c. Pengintegrasian *web service* dengan *database*.
d. Pengintegrasian Unity dengan *web service*.

2. Analisis dan Perancangan Sistem

Pada tahap ini dilakukan analisis dan pendefinisian kebutuhan sistem untuk mengetahui permasalahan yang sedang dihadapi. Selanjutnya, dirumuskan rancangan sistem yang dapat memberikan solusi terhadap permasalahan tersebut. Langkah yang dilakukan pada tahap ini adalah sebagai berikut.

a. Perancangan dan pemodelan basis data dalam bentuk model konseptual fisik pada permainan sosial *Card Warlock Saga*.
b. Perancangan *stored procedure* basis data pada permainan sosial *Card Warlock Saga*.
c. Perancangan arsitektur *web service*.

3. Implementasi

Pada tahap ini desain perangkat lunak diwujudkan dalam bentuk kode program. Tahap ini merealisasikan apa yang terdapat pada tahapan sebelumnya menjadi sebuah aplikasi yang sesuai dengan apa yang telah direncanakan, di antaranya sebagai berikut.

a. Implementasi pembuatan basis data dengan menggunakan sistem manajemen basis data relasional MySQL.
b. Implementasi pembuatan *stored procedure* dengan menggunakan bantuan kakas SQLyog.
c. Implementasi pembuatan *web service* dengan menggunakan bahasa pemrograman PHP serta pengintegrasian dengan Unity. Kakas bantu yang digunakan adalah NetBeans IDE 7.2.

4. Pengujian dan evaluasi

Pada tahap ini dilakukan pengujian perangkat lunak yang bertujuan untuk menemukan kesalahan-kesalahan dalam pengembangan aplikasi secara langsung pada perangkat bergerak Android agar dapat diperbaiki sesegera mungkin.

Pengujian dilakukan pada masing-masing fungsi yang ada pada *web service*. Adapun poin-poin pengujian yang dilakukan adalah sebagai berikut.

a. *Correctness*
   Pengujian ini merupakan uji coba beberapa skenario pada fungsi-fungsi yang ada pada *web service* untuk menguji apakah *output* yang dikeluarkan oleh sistem sudah sesuai dengan *input* yang diberikan, dan sesuai dengan kebutuhan.
b. *Integrity*
   Pengujian ini merupakan uji coba untuk membandingkan hasil *output* suatu fungsi pada *web service* yang diuji pada beberapa *layer* berbeda yaitu pada *browser*, *console*, dan

pada *game*. Uji coba ini dilakukan untuk menunjukkan bahwa integritas data sudah tercapai dengan adanya hasil *output* yang sama pada beberapa *layer* pengujian yang berbeda.

5. **Penyusunan buku tugas akhir**

   Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan tugas akhir.

1.6. Sistematika Penulisan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan
Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan tugas akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan tugas akhir.

Bab II Dasar Teori
Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan tugas akhir ini.

Bab III Analisis dan Perancangan Sistem
Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan data dan arsitektur sistem.

Bab IV Implementasi
Bab ini berisi implementasi dari perancangan perangkat lunak.

Bab V Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian hasil analisis kakas.

Bab VI Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan tugas akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada aplikasi ini.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan tugas akhir. Berikut uraian singkat mengenai poin-poin yang menjadi dasar teori dalam pembuatan tugas akhir ini.

2.1. MySQL

MySQL digunakan dalam perancangan dan implementasi sebuah basis data. MySQL merupakan sebuah *Relational Database Management System* (RDBMS), di mana pengguna dapat merancang, mengimplementasi, serta memanipulasi data dari *database* yang telah dibuatnya sendiri [1] .

2.1.1. *Stored Procedure*

Prosedur penyimpanan data atau biasa disebut dengan *stored procedure* adalah suatu kumpulan fungsi objek yang tersimpan dalam basis data dan dapat digunakan untuk menggantikan berbagai kumpulan perintah yang sering digunakan [2]. *Stored procedure* sangat berguna ketika pengembang aplikasi tidak ingin pengguna mengakses tabel pada basis data secara langsung (membatasi hak akses pengguna) dan mencatat operasi yang dilakukan sehingga resiko kerusakan data dapat diminimalisir.

2.2. *Web Service*

Web service merupakan sebuah antarmuka yang menggambarkan sekumpulan operasi fungsi-fungsi yang dapat diakses melalui sebuah *network*. *Web service* dideskripsikan dengan standar XML yang disebut dengan *service description*, yang mencakup isi dari *service* itu sendiri, yakni: format pesan, *input* dan *output* dari setiap fungsi, serta protokol yang digunakan. *Interface* dari sebuah *web service* menyembunyikan detail dari deskripsi *web service* itu sendiri, sehingga memungkinkan untuk digunakan secara terpisah dari

hardware, *platform*, maupun *software* yang berhubungan dengannya [3].

2.2.1. WSDL

Web Service Definiton Language (WSDL) merupakan sebuah format XML untuk menggambarkan sebuah *web service*, yang berisi tentang deskripsi dari *web service* tersebut. Operasi dan pesan pada *web service* dijelaskan secara abstrak pada WSDL [4].

2.2.2. SOAP

SOAP merupakan singkatan dari *Simple Object Access Protocol*. Aplikasi utama SOAP adalah komunikasi antar aplikasi (*application-to-application*). SOAP itu sendiri merupakan sebuah bentuk dokumen XML yang dapat disebut dengan "SOAP *Message"*. SOAP *message* inilah yang nantinya di-*transport* sepanjang *network*. SOAP *message* berjalan melalui HTTP SOAP *message* [5]. Beberapa contoh *tag* atau struktur pada SOAP antara lain seperti: *<soap:Body>*, *<soap:Envelope>*, *<soap:Header>*.

2.3. CodeIgniter

CodeIgniter adalah salah satu *framework* yang ada pada bahasa pemrograman PHP. *Framework* CodeIgniter menggunakan konsep MVC (*Model View Controller*) untuk membangun sebuah aplikasi *web*. CodeIgniter juga mempunyai beberapa jenis *library* di dalamnya, termasuk *nusoap_library* yang memungkinkan pengguna untuk merancang dan mengimplementasikan *web service* [3].

2.3.1. MVC

MVC (*Model View Controller*) merupakan sebuah konsep yang digunakan pada *framework* CodeIgniter. Pada konsep ini, terdapat tiga buah komponen yang masing-masing berhubungan satu sama lain [6], yaitu sebagai berikut.

a. *Model*
   Komponen *model* merepresentasikan sebuah komponen atau kelas yang konkrit, mencakup atribut kelas serta fungsi-fungsi yang ada pada kelas tersebut. Komponen *model* juga berkaitan dengan *database* pada *scope* ini. Komponen *model* hanya dapat berhubungan dengan komponen *controller* (pada *scope* MVC).
b. *View*
   Komponen *view* berhubungan dengan tampilan *web*. *View* mengatur apa saja yang akan ditampilkan yang sebelumnya telah diatur sedemikian rupa dari komponen *controller*, karena *view* tidak dapat mengakses *model* secara langsung melainkan melalui *controller*.
c. *Controller*
   Controller merupakan komponen penengah atau penghubung antara *model* dan *view*. Pengaturan pemanggilan *model* apa saja yang dipakai saat ini, serta tampilan atau *view* apa saja yang akan ditampilkan diatur pada komponen ini.

2.4. Social Game

Social Game adalah permainan yang dimainkan dengan teman-teman yang berada pada suatu jaringan sosial sehingga pemain bisa mengundang atau mengajak teman untuk melakukan aktivitas tertentu dalam game [7]. *Social Game* pada umumnya bersifat *asynchronous* di mana untuk memainkannya pemain tidak perlu *online* secara bersama–sama.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Bab ini membahas tahap analisis permasalahan dan perancangan dari sistem yang akan dibangun. Analisis permasalahan membahas permasalahan yang diangkat dalam pengerjaan tugas akhir. Analisis kebutuhan mencantumkan kebutuhan-kebutuhan yang diperlukan perangkat lunak. Selanjutnya dibahas mengenai perancangan sistem yang dibuat.

3.1. Analisis

Tahap analisis dibagi menjadi beberapa tahapan, yakni analisis permasalahan, deskripsi umum sistem, arsitektur perangkat lunak, arsitektur sistem dan penjelasan fungsi-fungsi yang ada pada *web service*.

3.1.1. Analisis Permasalahan

Permasalahan utama yang diangkat dalam pembuatan tugas akhir ini adalah bagaimana merancang dan mengimplementasikan *web service* dengan menggunakan bahasa pemrograman PHP yang dapat digunakan dalam permainan sosial perangkat bergerak *Card Warlock Saga*. Permasalahan berikutnya adalah bagaimana cara merancang dan mengimplementasikan *database* serta manajemen transaksi aturan main yang ada pada permainan *Card Warlock Saga* dengan menggunakan MySQL dan mengintegrasikannya dengan *web service* yang dibangun.

3.1.2. Deskripsi Umum *Game*

Card Warlock Saga merupakan permainan *Turn-Based Collectible Card*, yaitu permainan bergenre RPG yang menggunakan kartu sebagai mekanisme pertarungan permainan. Pemain memiliki satu *hero* dan set kartu sebagai mekanisme pertarungan. Keseluruhan sistem pada permainan ini digambarkan dengan *block* diagram pada Gambar 9.1 dan Gambar 9.2. Pada

block diagram ini terdapat beberapa modul atau fungsi yang dikategorikan menjadi 3 modul utama, yakni: modul *battle*, modul *shopping*, dan modul *social*. Ketiga modul tersebut mempunyai fungsi masing-masing, yang nantinya akan menuju ke *web service*. Modul *web service* ditandai dengan kotak warna merah pada *block* diagram.

Permainan ini dibangun dengan menggunakan teknologi Unity 2D dan menggunakan sistem manajemen basis data pada bagian *server* untuk menyimpan data-data yang ada, seperti data kartu dan pemain, data transaksi *trading card*, serta data *record* pertarungan setiap pemain. Setiap proses transaksi data dari klien menuju *server* dan sebaliknya, akan diarahkan ke *web service* terlebih dahulu. Untuk bahasa pemrograman yang dipakai pada pembangunan *web service* adalah PHP, dikarenakan PHP lebih dikenal dan *familiar* di kalangan masyarakat dan *scope* PHP pada kategori permainan cukup luas. PHP juga sudah umum berhubungan dengan manajemen basis data MySQL dan dipermudah dengan adanya *framework* CodeIgniter, seperti yang digunakan pada permainan *Card Warlock Saga*.

3.1.3. Arsitektur Perangkat Lunak

Web service pada permainan *Card Warlock Saga* menyediakan fungsi-fungsi yang dibutuhkan pada permainan, mulai dari fungsi-fungsi pada modul *battle*, *shopping*, *social*, serta fungsi-fungsi aturan main yang ada. Pengguna aplikasi permainan ini dapat mengakses fungsi-fungsi yang ada pada *web service* melalui HTTP *request* dengan mengirimkan nama fungsi apa yang diperlukan disertai dengan parameter yang dibutuhkan.

Secara umum, terdapat 2 jenis fungsi yang dapat diakses oleh pengguna melalui *web service* ini. Fungsi tersebut adalah *set* dan yang kedua adalah *get*. Fungsi *set* meliputi pengaksesan fungsi yang berhubungan dengan perubahan data pada *database*, yakni: *insert*, *update*, dan *delete* data. Sedangkan fungsi *get* adalah

fungsi yang berguna untuk mengambil data dari *database* (*select* data).

Pada Gambar 9.3 dijabarkan mengenai arsitektur perangkat lunak pada *web service* ini. Pengguna melakukan *request* data dari aplikasi kepada *web service*, yang kemudian akan diproses oleh fungsi-fungsi yang ada pada *web service* dan diteruskan ke *database*. Kemudian *web service* akan mengembalikan hasil *query* fungsi yang telah dijalankan, baik fungsi *get* maupun fungsi *set*, kepada aplikasi dalam bentuk data yang telah diserialisasi dalam bentuk XML untuk kemudian diproses pada saat permainan berlangsung.

3.1.4. Arsitektur Sistem

Berikut dijelaskan mengenai cara kerja sistem pada *web service* permainan ini dalam bentuk *activity diagram* yang dapat dilihat Pada Gambar 9.4. Pengguna aplikasi permainan ini memasuki sistem pada *web service* dengan mengirimkan *request* dari aplikasi kepada *web service*. Sistem kemudian akan memanggil fungsi yang diperlukan dan memproses *input* dari pengguna pada fungsi tersebut. Apabila *input* memenuhi syarat maka data akan dilanjutkan ke *database* untuk diproses, namun apabila terjadi kesalahan *input* maka sistem akan mengembalikan *exception* kepada pengguna aplikasi. Data yang telah dikirim ke *database* kemudian akan diproses dan dilakukan *query*. Hasil *query* akan dikembalikan kepada *web service*. Sistem dari *web service* kemudian akan mengembalikan hasil *query* kepada pengguna aplikasi dalam bentuk data yang telah diserialisasi dalam bentuk XML, yang kemudian akan dibaca dan diproses oleh aplikasi permainan.

3.1.5. Fungsi-Fungsi pada *Card Warlock Saga*

Telah disebutkan sebelumnya bahwa pada permainan *Card Warlock Saga* terdapat beberapa fungsi yang dikategorikan

menjadi 3 kategori pada *web service*, yakni: modul *battle*, modul *shopping*, dan modul *social*. Selain ketiga modul tersebut, juga terdapat satu kategori fungsi lain pada *web service* yaitu fungsi aturan main. Untuk penamaan fungsi pada *web service*, mempunyai beberapa kriteria sebagai berikut.

- Nama fungsi minimal terdiri atas 1 kata.
- Kata pertama merupakan kata kerja yang merepresentasikan isi fungsi. Untuk kata kerja yang digunakan antara lain adalah: *get* (mengambil data untuk diproses pada permainan), *show* (mengambil data untuk ditampilkan dalam permainan), *update* (melakukan *update* data), *insert* (memasukkan data baru), *send* (mengirimkan data dari klien ke *server*), *remove* (menghapus data), *buy* dan *sell* (pembelian dan penjualan kartu), *accept* dan *decline* (menerima dan menolak permintaan atau *request*), *edit* (melakukan perubahan data), dan *share* (melakukan proses *share* ke Facebook).
- Kata yang mengikuti kata pertama merupakan kata benda yang merepresentasikan nama data yang akan diolah pada fungsi tersebut.
- Setiap kata dipisahkan dengan *underscore* (_). Misal untuk fungsi dengan nama *get player profile* ditulis dengan *get_player_profile*().
- Setiap kata ditulis dengan *lower* case. Misal untuk fungsi *send trade request* ditulis dengan *send_trade_request*() tanpa ada huruf *upper case*.

Berikut diuraikan mengenai fungsi-fungsi apa saja yang ada pada setiap kategori tersebut.

3.1.5.1. Fungsi *Battle*

Pada kategori ini, terdapat beberapa fungsi yang terkait dengan adanya *battle* pada permainan, baik *battle* antar pemain maupun

battle antara pemain dengan kecerdasan buatan. Tabel 3.1 berikut adalah rincian fungsi-fungsi yang terkait dengan *battle*.

Tabel 3.1. Rincian fungsi pada kategori *battle*

Nama Fungsi	Jenis	Keterangan
`send_battle_result()`	*set*	Mengirimkan hasil *battle* antar pemain dan meng-*update* data *battle* pemain.
`show_player_ranking()`	*get*	Menampilkan *ranking* 5 pemain terbaik dengan jumlah *win* terbanyak.
`show_battle_rank()`	*get*	Menampilkan data *battle detail* antara pemain satu dengan pemain lain.
`update_data()`	*set*	Meng-*update* data pemain setelah pemain selesai melakukan *battle* dengan kecerdasan buatan.
`get_cards()`	*get*	Mengambil data kartu pemain, baik kartu yang ada pada *trunk*, maupun kartu yang ada pada *deck*.
`insert_to_deck()`	*set*	Memasukkan kartu yang diinginkan ke dalam *deck* pemain.
`clear_deck()`	*set*	Mengosongkan *deck* pemain dan mengembalikan kartu ke dalam *trunk*.

Nama Fungsi	Jenis	Keterangan
`get_player_quest()`	*get*	Mengambil data *quest* pemain.
`get_battle_list()`	*get*	Mengambil *list* data *battle* pada satu *dungeon*.
`get_monster_list()`	*get*	Mengambil *list* data *monster* pada satu *battle*.

Pada Tabel 3.1 ditunjukkan rincian fungsi-fungsi yang termasuk dalam kategori *battle*, mencakup *battle* antar pemain dan *battle* antara pemain dengan kecerdasan buatan, data *quest* pemain serta manajemen kartu pemain yang disertai dengan jenis masing-masing fungsi. Sedangkan Tabel 3.2 berikut ini adalah rincian dari *input* dan *output* masing-masing fungsi pada kategori *battle*.

Tabel 3.2. Rincian *input* dan *output* fungsi pada kategori *battle*

Nama Fungsi	*Input Parameter*	*Return Value*
`send_battle_result()`	• *PlayerWinner (string)* • *PlayerLoser (string)*	*String* berupa "*Battle Result Updated*".
`show_player_ranking()`	-	Data *ranking* pemain sebagai berikut. - Nama pemain (*string*). - Poin pemain (*int*).

Nama Fungsi	*Input Parameter*	*Return Value*
`show_battle_rank()`	• *PlayerName (string)*	Data *battle detail* antar pemain sebagai berikut. - Nama pemain lawan (*string*). - Poin menang (*int*). - Poin kalah (*int*).
`update_data()`	• *PlayerName (string)* • *GoldEarned (int)* • *XPEarned (int)*	*String* berupa "*Data Updated*".
`get_cards()`	• *FunctionName (string)* • *PlayerName (string)*	Data kartu yang dimiliki oleh pemain sesuai nama fungsinya (*deck* atau *trunk*) dengan rincian sebagai berikut. - Nama kartu (*string*). - Jumlah kartu (*int*).
`insert_to_deck()`	• *PlayerName*	*String* berupa "*Card*

Nama Fungsi	*Input Parameter*	*Return Value*
	(string) • *CardName (string)* • *Quantity (int)*	*has been added to deck*".
`clear_deck()`	• *PlayerName*	*String* berupa "*Deck is cleared*".
`get_player_quest()`	• *PlayerName (string)* • *DungeonID (string)*	Rincian data *quest* pemain pada *dungeon* tertentu, dengan rincian sebagai berikut. - ID *quest* (*string*). - Keterangan aktif tidaknya *quest* tersebut (*int*). - Keterangan telah selesai atau tidaknya *quest* tersebut (*int*).
`get_battle_list()`	• *PlayerName (string)* • *DungeonID (string)*	Data *sub dungeon* pada *dungeon* yang dipilih, dengan rincian sebagai berikut. - ID *sub*

Nama Fungsi	*Input Parameter*	*Return Value*
		dungeon (*string*). - Keterangan aktif tidaknya *sub dungeon* tersebut (*int*).
`get_monster_list()`	• *BattleID (string)*	Data *monster list* pada *battle* yang dipilih, dengan rincian sebagai berikut. - Nama *monster* (*string*). - Jumlah *monster* (*int*).

Pada Tabel 3.2 ditunjukkan mengenai rincian *parameter input* serta *output* dari setiap fungsi yang ada pada kategori *battle*.

3.1.5.2. Fungsi *Shopping*

Fungsi-fungsi pada kategori ini berhubungan dengan data transaksi *shopping*, yakni pembelian dan penjualan *card* pada *shop*. Tabel 3.3 berikut adalah rincian fungsi-fungsi yang ada pada kategori ini.

Tabel 3.3. Rincian fungsi pada kategori *shopping*

Nama Fungsi	Jenis	Keterangan
`get_shop_card()`	*get*	Menampilkan data kartu yang disediakan oleh *shop*.
`buy_card()`	*set*	Membeli kartu yang diinginkan dan memasukkan kartu tersebut ke dalam *trunk*.
`sell_card()`	*set*	Menjual kartu yang dikehendaki kepada *shop*.

Pada Tabel 3.3 ditunjukkan rincian fungsi-fungsi yang ada pada kategori *shopping* beserta jenis fungsinya. Sedangkan Tabel 3.4 berikut ini adalah rincian dari *input* dan *output* masing-masing fungsi pada kategori *shopping*.

Tabel 3.4. Rincian *input* dan *output* fungsi pada kategori *shopping*

Nama Fungsi	*Input Parameter*	*Return Value*
`get_shop_card()`	-	Data kartu yang disediakan oleh *shop*, dengan rincian sebagai berikut. - Nama kartu (*string*). - Harga kartu (*int*).

Nama Fungsi	*Input Parameter*	*Return Value*
`buy_card()`	• *PlayerName (string)* • *CardName (string)* • *Quantity (int)*	Keterangan berhasil tidaknya proses transaksi pembelian kartu, dengan rincian sebagai berikut. - *String* berupa "*Card Received*" apabila transaksi berhasil. - *String* berupa "*Not Enough Money*" apabila uang tidak mencukupi.
`sell_card()`	• *PlayerName (string)* • *CardName (string)* • *Quantity (int)*	*String* berupa "*Card Removed*".

Pada Tabel 3.4 ditunjukkan rincian *parameter input* serta *output* dari setiap fungsi yang ada pada kategori *shopping*.

3.1.5.3. Fungsi *Social*

Fungsi-fungsi pada kategori ini berhubungan dengan fitur sosial pada *game*, yakni berhubungan antara pemain satu dengan

pemain lainnya. Tabel 3.5 berikut ini merupakan rincian fungsi-fungsi pada kategori ini.

Tabel 3.5. Rincian fungsi pada kategori *social*

Nama Fungsi	Jenis	Keterangan
`do_request()`	*set*	Melakukan *request* kepada pemain lain. *Request* pada fungsi ini berupa *friend request*. *Request* dibedakan menjadi 2 macam, yakni *send request* dan *accept request*.
`get_friend_request_list()`	*get*	Mengambil data *friend request* yang dikirim kepada pemain.
`get_avatar()`	*get*	Mengambil data *avatar* yang dipakai oleh pemain saat ini.
`edit_avatar()`	*set*	Meng-*update* data *avatar* pemain.
`send_message()`	*set*	Mengirim pesan kepada pemain lain.
`get_messages()`	*get*	Mengambil data pesan masuk pada pemain.
`send_trade_request()`	*set*	Mengirim permintaan pertukaran kartu antar pemain.
`get_trade_request()`	*get*	Mengambil data *trading request* pada pemain.
`accept_trade_request()`	*set*	Menerima permintaan

Nama Fungsi	Jenis	Keterangan
		pertukaran kartu dan meng-*update* data kartu pemain.
`remove_friend()`	*set*	Menghapus teman dari *friend list*.
`get_profile()`	*get*	Mengambil data profil pemain.
`get_partial_profile()`	*get*	Mengambil data pemain secara umum. Fungsi ini digunakan pada fitur pencarian teman pada *game*.
`get_friend_list()`	*get*	Mengambil data teman dari seorang pemain.
`share_gift()`	*set*	Mengirim *gift* yang dapat diambil oleh teman.

Pada Tabel 3.5 ditunjukkan rincian fungsi-fungsi yang ada pada kategori *social*. Fungsi-fungsi pada kategori ini berhubungan dengan pemain lain yang berupa *request*, data *player message* serta data *avatar* pemain yang dapat dilihat oleh pemain lain. Sedangkan Tabel 3.6 berikut ini adalah rincian dari *input* dan *output* masing-masing fungsi pada kategori *social*.

Tabel 3.6. Rincian *input* dan *output* fungsi pada kategori *social*

Nama Fungsi	*Input Parameter*	*Return Value*
`do_request()`	• *RequestType (string)* • *PlayerName*	Keterangan berhasil tidaknya proses pengiriman atau

Nama Fungsi	*Input Parameter*	*Return Value*
	(string) • *FriendName (string)*	penerimaan *request*, dengan rincian sebagai berikut. - *String* berupa "*Request Sent*" untuk pengiriman *request.* - *String* berupa "*Request Accepted*" untuk penerimaan *request.* - *String* berupa "*Request Declined*" untuk penolakan *request.*
`get_friend_request_list()`	• *PlayerName (string)*	Data *friend request* yang dikirimkan kepada pemain, dengan rincian dengan sebagai berikut. - Nama pemain (*string*). - *Job* pemain (*string*). - *Rank* pemain (*string*). - *Level* pemain (*int*).

Nama Fungsi	*Input Parameter*	*Return Value*
`get_avatar()`	• *PlayerName (string)*	Data *avatar* pemain, dengan rincian sebagai berikut. - Bagian / *slot avatar* (*string*). - Nama *part avatar* (*string*).
`edit_avatar()`	• *PlayerName (string)* • *AvatarName (string)*	*String* berupa "*Avatar Updated*".
`send_message()`	• *PlayerName (string)* • *Subject (string)* • *Message (string)*	*String* berupa "*Message Sent*".
`get_messages()`	• *PlayerName (string)*	Data pesan pada *inbox* pemain, dengan rincian sebagai berikut. - Nama pengirim (*string*). - Subjek pesan (*string*). - Isi pesan (*string*).
`send_trade_request()`	• *Trade Request Object (encoded xml data)*	*String* berupa "*Trade Request Sent*".

Nama Fungsi	*Input Parameter*	*Return Value*
`get_trade_request()`	• *PlayerName (string)*	Data *trading request* yang diterima oleh pemain, dengan rincian sebagai berikut. - Nama pengirim (*string*). - ID *trade request* (*int*).
`accept_trade_request()`	• *RequestID (int)*	*String* berupa "*Trade Request Accepted*".
`remove_friend()`	• *PlayerName (string)* • *FriendName (sting)*	*String* berupa "*Friend Removed*".
`get_profile()`	• *PlayerName (string)*	Data pemain, dengan rincian sebagai berikut. - Nama pemain (*string*). - *Job* pemain (*string*). - *Rank* pemain (*string*). - *Level* pemain (*int*). - Jumlah uang pemain (*int*). - Jumlah poin HP, SP, XP dan

Nama Fungsi	*Input Parameter*	*Return Value*
		DP pemain (*int*). - Jumlah poin menang dan kalah pemain (*int*).
`get_partial_profile()`	• *PlayerName (string)*	Data pemain secara umum, dengan rincian sebagai berikut. - Nama pemain (*string*). - *Job* pemain (*string*). - *Rank* pemain (*string*). - *Level* pemain (*int*).
`get_friend_list()`	• *PlayerName (string)*	Data *friend list* dari seorang pemain, dengan rincian sebagai berikut. - Nama pemain (*string*). - *Job* pemain (*string*). - *Rank* pemain (*string*). - *Level* pemain (*int*).

Nama Fungsi	*Input Parameter*	*Return Value*
`share_gift()`	• *PlayerName (string)*	*String* berupa "*Gift Shared*".

Pada Tabel 3.6 ditunjukkan rincian *parameter input* serta *output* dari setiap fungsi yang ada pada kategori *shopping*.

3.1.5.4. Fungsi Aturan Main

Untuk aturan main pada permainan *Card Warlock Saga* diuraikan sebagai berikut.

- *Battle*
  - Pemain dapat memiliki banyak kartu. Pemain dapat memasukkan kartu apa saja yang diinginkan ke dalam *deck* dan sejumlah berapapun asalkan tidak melebihi batas kuota yang dituliskan dalam bentuk *deck point*, di mana setiap kartu mempunyai nilai *deck point* yang berbeda-beda.
  - Pemain dapat masuk ke *dungeon* dan menyelesaikan *quest* dengan bertarung dengan *monster-monster* yang ada. Namun pemain hanya dapat memasuki *dungeon* sebanyak jumlah energi yang dimilikinya pada hari tersebut. Setiap *quest* yang terselesaikan akan membuka *quest* selanjutnya.
  - Pemain dapat melakukan serangan terhadap musuh pada saat *battle* dengan memilih kartu yang ada pada *deck*. Namun pemain hanya dapat melakukan aksi sebanyak batas aksi yang diijinkan yang dituliskan dengan *soul point.* Setiap kartu mempunyai nilai *soul point* yang berbeda-beda.
  - Pemain mendapatkan *gold* dan XP setiap menyelesaikan sebuah *quest*. Pemain dapat naik *level* dan *rank* apabila XP telah melebihi batas maksimal saat ini.

- Pemain dapat menantang pemain lain dan bertarung dengannya. Hasil pertarungan akan dicatat sebagai poin menang atau kalah setiap pemain.
- Pemain dapat melihat rekor pemain terbaik dengan jumlah poin kemenangan tertinggi.

- *Shopping*
  - Pemain dapat melihat kartu apa saja yang dijual di toko.
  - Pemain dapat memilih kartu yang diinginkan dan membeli kartu tersebut di toko.
  - Pemain dapat menjual kartu yang dimilikinya.

- *Social*
  - Pemain dapat mengirim permintaan pertemanan kepada pemain lain.
  - Pemain yang dikirimi permintaan pertemanan dapat menerima maupun menolak permintaan pertemanan.
  - Pemain dapat melihat daftar teman yang dimilikinya.
  - Pemain dapat menghapus pemain dari daftar teman.
  - Pemain dapat melihat profil dirinya yang berisi data pemain.
  - Pemain dapat melihat profil teman.
  - Pemain dapat mengirim pesan ke *inbox* pesan teman.
  - Pemain dapat mengirim permintaan pertukaran kartu kepada teman.
  - Pemain dapat mengirim *gift* melalui Facebook kepada teman dengan batasan 1 *gift* per hari.
  - Pemain dapat merubah penampilan wajah pada *avatar* dirinya.

- Umum
  - Pemain dapat melakukan proses *login* ke dalam permainan.
  - Pemain dapat mendaftarkan diri ke dalam permainan lewat fungsi *register*.

- Pemain dapat menambahkan *building* dan melakukan *upgrade* terhadap *building* tersebut.
- Pemain dapat mengambil *gift* yang telah dibagikan lewat Facebook.

Setiap aturan main yang telah dituliskan di atas terbagi-bagi dan diimplementasikan pada fungsi *battle*, *shopping*, dan *social*. Untuk rincian fungsi aturan main pada kategori umum, diuraikan pada Tabel 3.7 berikut ini.

Tabel 3.7. Rincian fungsi pada kategori aturan main

Nama Fungsi	Jenis	Keterangan
`get_xp_pool ()`	*get*	Mengambil data *experience pool* dari pemain berdasarkan *level* pemain saat ini.
`level_up()`	*get*	Meng-*update rank* pemain berdasarkan *level* pemain saat ini.
`calculate_data()`	*set*	Menghitung perubahan data pemain saat *pre-battle* dengan kecerdasan buatan. Fungsi ini juga menentukan apakah pemain dapat melakukan *level up* berdasarkan XP yang dimilikinya.
`get_building()`	*get*	Mengambil data *building* yang dimiliki oleh pemain.
`update_building()`	*set*	Meng-*update* data *building* pemain.

Nama Fungsi	Jenis	Keterangan
`get_name_by_fb()`	*get*	Mengambil data nama pemain dari Facebook ID yang dimiliki.
`register()`	*set*	Mendaftarkan pemain baru pada *game*.
`login()`	*set*	Melakukan proses *login* pemain ke dalam permainan.

Pada Tabel 3.7 disebutkan satu per satu mengenai fungsi-fungsi yang ada pada kategori aturan main *game*, disertai dengan jenis fungsinya. Sedangkan Tabel 3.8 berikut adalah rincian dari *input* dan *output* masing-masing fungsi pada kategori aturan main.

Tabel 3.8. Rincian *input* dan *output* fungsi pada kategori aturan main

Nama Fungsi	*Input Parameter*	*Return Value*
`get_xp_pool()`	• *PlayerRank (string)* • *PlayerLevel (int)*	Data XP *pool* pemain (*int*).
`level_up()`	• *PlayerRank (string)*	Data *rank* pemain (*string*).
`calculate_data()`	• *PlayerName (string)* • *GoldEarned (int)* • *XPEarned*	Keterangan perubahan data pada pemain, dengan rincian sebagai berikut. - Poin XP pemain

Nama Fungsi	*Input Parameter*	*Return Value*
	(int)	(*int*). - Jumlah *gold* pemain (*int*). - Data *rank* pemain (*string*). - Data *level* pemain (*int*).
`get_building()`	• *PlayerName (string)*	Data *building* yang dimiliki oleh pemain, dengan rincian sebagai berikut. - Nama *building* (*string*). - Jumlah poin XP maksimal dan poin XP saat ini (*int*).
`update_building()`	• *BuildingName (string)* • *Building Stats (int)* • *ProductQuantity (int)*	*String* berupa "*Building Updated*".
`get_name_by_fb()`	• *FB_ID (string)*	Data nama pemain (*string*).
`register()`	• *PlayerName (string)* • *Password (string)*	*String* berupa "*Registration Success*".

Nama Fungsi	*Input Parameter*	*Return Value*
	• *Job (string)*	
`login()`	• *PlayerName (string)* • *Password (string)*	- *String* berupa "*Login Success*" apabila berhasil. - *String* berupa "*Login Failed*" apabila gagal.

Pada Tabel 3.8ditunjukkan rincian *parameter input* serta *output* dari setiap fungsi yang ada pada kategori *shopping*.

3.2. Perancangan Sistem

Penjelasan tahap perancangan dibagi menjadi beberapa bagian yaitu perancangan tabel dan *database*, perancangan *stored procedure* dan perancangan kelas pada program PHP.

3.2.1. Perancangan Basis Data

Pada sub-bab ini, dijelaskan tentang perancangan basis data yang digunakan. Perancangan pada sub-bab ini dibagi menjadi 2 bagian, yakni *physical data model* dan perancangan tabel.

3.2.1.1. Physical Data Model

Berikut rancangan tabel pada *database* beserta relasi setiap tabelnya yang digambarkan dalam bentuk *physical data model* (PDM) yang dapat dilihat pada Gambar 9.5, di mana gambar tersebut menunjukkan perancangan basis data pada permainan ini dalam bentuk tabel beserta relasinya antara satu tabel dengan yang lainnya.

3.2.1.2. Perancangan Tabel

Berikut dijelaskan mengenai perancangan setiap tabel yang ada pada *database* permainan *Card Warlock Saga*.

3.2.1.2.1 Tabel `players`

Tabel ini berisi data-data pemain. Rincian atribut yang dimiliki oleh tabel `players` dapat dilihat pada Tabel 3.9.

Tabel 3.9. Rincian atribut tabel `players`

Nama Atribut	Tipe Data	Keterangan
`PlayerID`	*Varchar*(50)	*Primary key* tabel. PlayerID berisi nama *username* pemain.
`FB_ID`	*Varchar*(50)	Berisi Facebook ID pemain apabila pemain mendaftar lewat Facebook.
`Email`	*Varchar*(50)	Berisi alamat *e-mail* pemain.
`Gold`	*Int*	Menyimpan jumlah uang pemain saat ini.
`MaxHP`	*Int*	Menyimpan nilai kapasitas *health point* (HP) pemain.
`MaxSP`	*Int*	Menyimpan nilai kapasitas *soul point* (SP) pemain saat ini.
`MaxDP`	*Int*	Menyimpan nilai kapasitas *deck point* (DP) pemain saat

Nama Atribut	Tipe Data	Keterangan
		ini.
`Battle_Won`	*Int*	Menyimpan jumlah *battle* yang telah dimenangkan pemain terhadap pemain lain.
`Battle_Lost`	*Int*	Menyimpan jumlah *battle* di mana pemain kalah terhadap pemain lain.
`XP`	*Int*	Menyimpan nilai *experience point* (XP) pemain saat ini.
`Energy`	*Int*	Menyimpan nilai energi yang dimiliki pemain saat ini.
`Job`	*Varchar*(15)	Menyimpan *job* yang dimiliki oleh pemain.
`Rank`	*Varchar*(1)	Menyimpan tingkatan *rank* pemain saat ini.
`Level`	*Int*	Menyimpan tingkatan *level* pemain saat ini.
`Gender`	*Varchar*(10)	Menyimpan jenis kelamin pemain.
`Password`	*Varchar*(25)	Menyimpan *password* dari pemain.

Pada Tabel 3.9 dijelaskan tentang keterangan masing-masing atribut pada tabel `player` beserta tipe data setiap atributnya. Tabel `players` merupakan tabel *master* yang paling banyak berelasi dengan tabel-tabel lain.

3.2.1.2.2 Tabel `player_friend`

Tabel `player_friend` menunjukkan hubungan pertemanan antara pemain satu dengan pemain lain. Rincian atribut pada tabel ini dapat dilihat pada Tabel 3.10.

Tabel 3.10. Rincian atribut tabel `player_friend`

Nama Atribut	Tipe Data	Keterangan
`FriendID`	*Int*	*Primary key* tabel.
`PlayerOne`	*Varchar*(50)	*Foreign key* dari tabel *players*. Berisi ID pemain pertama.
`PlayerTwo`	*Varchar*(50)	*Foreign key* dari tabel *players*. Berisi ID pemain kedua.

Pada Tabel 3.10 dijelaskan tentang keterangan masing-masing atribut pada tabel `player_friend` beserta tipe datanya masing-masing. Tabel `player_friend` merupakan bentuk relasi *many-to-many* antara tabel `players` dengan tabel `players` itu sendiri.

3.2.1.2.3 Tabel `player_message`

Tabel `player_message` menyimpan data pengiriman pesan antar pemain. Rincian atribut pada tabel ini dapat dilihat pada Tabel 3.11.

Tabel 3.11. Rincian atribut tabel `player_message`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Int*	*Primary key* tabel.
`PlayerSender`	*Varchar*(50)	*Foreign key* dari tabel *players*. Berisi ID pemain pengirim pesan.
`PlayerReceiver`	*Varchar*(50)	*Foreign key* dari tabel *players*. Berisi ID pemain penerima pesan.
`Subject`	*Text*	Berisi *subject* pesan.
`Message`	*Text*	Isi pesan yang dikirim.

Pada Tabel 3.11 dijelaskan mengenai keterangan masing-masing atribut pada tabel `player_message` beserta tipe datanya. Tabel `player_message` mempunyai hubungan *one-to-many* dengan tabel `players`.

3.2.1.2.4 Tabel `player_request`

Tabel `player_request` menyimpan data *request* yang dilakukan antar pemain. *Request* pada konteks ini adalah *request* pertemanan dan *request* energi. Rincian mengenai atribut-atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.12.

Tabel 3.12. Rincian atribut tabel `player_request`

Nama Atribut	Tipe Data	Keterangan
`RequestID`	*Int*	*Primary key* tabel.

Nama Atribut	Tipe Data	Keterangan
`SenderPlayer`	*Varchar*(50)	*Foreign key* dari tabel *players*. Berisi ID pemain pengirim *request*.
`RequestedPlayer`	*Varchar*(50)	*Foreign key* dari tabel *players*. Berisi ID pemain penerima *request*.
`RequestTime`	*DateTime*	Berisi waktu pengiriman *request*.
`Information`	*Varchar*(100)	Berisi informasi mengenai *request* yang dikirim.

Pada Tabel 3.12 dijelaskan mengenai keterangan masing-masing atribut pada tabel `player_request` beserta tipe datanya. Tabel `player_request` mempunyai hubungan *one-to-many* dengan tabel `players`.

3.2.1.2.5 Tabel `setting`

Tabel `setting` menyimpan informasi mengenai data pemain terkait pengaturan batasan dalam proses *share gift* dan pengaturan *claim product* pada *building*. Rincian mengenai masing-masing atribut pada tabel ini dapat dilihat pada Tabel 3.13.

Tabel 3.13. Rincian atribut tabel `setting`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Int*	*Primary key* tabel.
`PlayerID`	*Varchar(50)*	*Foreign key* dari tabel *players*. Berisi ID pemain.
`ShareGiftState`	*Int*	Berisi batasan *share* yang dapat dilakukan oleh pemain.
`AltarClaimState`	*Int*	Berisi batasan *product claim* yang dapat dilakukan oleh pemain pada *building altar*.
`MineClaimState`	*Int*	Berisi batasan *product claim* yang dapat dilakukan oleh pemain pada *building mine*.
`YggdrasilClaimState`	*Int*	Berisi batasan *product claim* yang dapat dilakukan oleh pemain pada *building yggdrasil*.

Pada Tabel 3.13 dijelaskan mengenai keterangan masing-masing atribut pada tabel `setting` beserta tipe datanya masing-masing.

Tabel `setting` hanya berelasi dengan tabel `players` dengan hubungan *one-to-many*.

3.2.1.2.6 Tabel `monster`

Tabel ini berisi data tentang `monster` yang ada pada permainan. Rincian masing-masing atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.14.

Tabel 3.14. Rincian atribut tabel `monster`

Nama Atribut	Tipe Data	Keterangan
`MonsterID`	*Varchar*(10)	*Primary key* tabel.
`Name`	*Varchar*(100)	Berisi nama *monster*.
`HP`	*Int*	Berisi nilai kapasitas *health point* pada *monster*.
`AttackPoint`	*Int*	Berisi nilai serangan dari *monster*.
`Element`	*Varchar*(25)	Berisi informasi mengenai elemen yang dimiliki *monster*.
`GoldReward`	*Int*	Berisi informasi mengenai jumlah uang yang didapat oleh pemain apabila mengalahkan *monster*.
`XPReward`	*Int*	Berisi informasi mengenai jumlah XP

Nama Atribut	Tipe Data	Keterangan
		yang didapat oleh pemain apabila mengalahkan *monster*.
`DroppableItem`	*Varchar*(50)	Berisi informasi mengenai data kartu yang dapat diperoleh oleh pemain setelah mengalahkan *monster*.
`Information`	*Text*	Deskripsi *monster*.

Pada Tabel 3.14 dijelaskan mengenai keterangan masing-masing atribut pada tabel `monster` beserta tipe datanya masing-masing. Tabel `monster` merupakan sebuah tabel *master*.

3.2.1.2.7 Tabel `dungeon`

Tabel `dungeon` berisi informasi mengenai *dungeon* yang ada pada permainan. Rincian mengenai atribut-atribut yang ada pada tabel ini dapat dilihat padaTabel 3.15.

Tabel 3.15. Rincian atribut tabel `dungeon`

Nama Atribut	Tipe Data	Keterangan
`DungeonID`	*Varchar*(10)	*Primary key* tabel.
`DungeonElement`	*Varchar*(50)	Berisi nama elemen pada *dungeon*.
`MapName`	*Varchar*(20)	Berisi nama lokasi *map* setiap *dungeon*.

Pada Tabel 3.15 dijelaskan mengenai keterangan masing-masing atribut pada tabel `dungeon` beserta tipe datanya masing-masing. Tabel `dungeon` merupakan sebuah tabel *master*.

3.2.1.2.8 Tabel `quest_battle`

Tabel `quest_battle` berisi mengenai *quest* atau *battle* yang ada pada setiap *dungeon*. Rincian atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.16.

Tabel 3.16. Rincian atribut tabel `quest_battle`

Nama Atribut	Tipe Data	Keterangan
`BattleID`	*Varchar*(10)	*Primary key* tabel.
`DungeonID`	*Varchar*(10)	*Foreign key* dari tabel *dungeon*. Menunjukkan *dungeon* tempat *battle* atau *quest* ini berada.

Pada Tabel 3.16 dijelaskan mengenai keterangan masing-masing atribut pada tabel `quest_battle` beserta tipe datanya masing-masing. Tabel `quest_battle` mempunyai hubungan *one-to-many* dengan tabel `dungeon`.

3.2.1.2.9 Tabel `battle_monster`

Tabel `battle_monster` berisi informasi mengenai *monster* apa saja yang ada pada setiap *quest* atau *battle*. Rincian mengenai setiap atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.17.

Tabel 3.17. Rincian atribut tabel `battle_monster`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Int*	*Primary key* tabel.
`BattleID`	*Varchar*(10)	*Foreign key* dari tabel *quest_battle*. Menunjukkan *battle* atau *quest* yang dimaksud.
`MonsterID`	*Varchar*(10)	*Foreign key* dari tabel *monster*. Menunjukkan *monster* apa saja yang ada pada *quest* atau *battle* yang dimaksud.

Pada Tabel 3.17 dijelaskan mengenai keterangan masing-masing atribut pada tabel `battle_monster` beserta tipe datanya masing-masing. Tabel `battle_monster` merupakan sebuah *junction table* yang merepresentasikan hubungan *many-to-many* antara tabel `monster` dengan tabel `quest_battle`.

3.2.1.2.10 Tabel `player_quest`

Tabel `player_quest` berisi informasi mengenai *quest* dari setiap pemain. Rincian atribut-atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.18.

Tabel 3.18. Rincian atribut tabel `player_quest`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Varchar*(15)	*Primary key* tabel.
`PlayerID`	*Varchar*(50)	*Foreign key* dari tabel

Nama Atribut	Tipe Data	Keterangan
		players. Menunjukkan ID pemain.
`BattleID`	*Varchar*(10)	*Foreign key* dari tabel *quest_battle.* Menunjukkan ID *quest* pemain.
`IsClear`	*Boolean*	Menunjukkan apakah *quest* sudah pernah diselesaikan atau belum.
`IsActive`	*Boolean*	Menunjukkan apakah sebuah *quest* sudah *available* atau belum.

Pada Tabel 3.18 dijelaskan mengenai keterangan masing-masing atribut pada tabel `player_quest` beserta tipe datanya masing-masing. Tabel `player_quest` merupakan sebuah *junction table* yang menghubungkan relasi *many-to-many* antara tabel `players` dengan tabel `quest_battle`.

3.2.1.2.11 Tabel `battle_detail`

Tabel `battle_detail` menyimpan informasi mengenai jumlah *battle* yang telah dilakukan antar pemain beserta informasi menang atau kalahnya pemain. Rincian masing-masing atribut pada tabel ini dapat dilihat pada Tabel 3.19.

Tabel 3.19. Rincian atribut tabel `battle_detail`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Int*	*Primary key* tabel.

Nama Atribut	Tipe Data	Keterangan
PlayerOne	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain pertama.
PlayerTwo	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain kedua.
Win	*Int*	Menunjukkan jumlah kemenangan pemain pertama terhadap pemain kedua.
Lose	*Int*	Menunjukkan jumlah kekalahan pemain pertama terhadap pemain kedua.

Pada Tabel 3.19 dijelaskan mengenai keterangan masing-masing atribut pada tabel `battle_detail` beserta tipe datanya masing-masing. Tabel *battle_detail* merupakan sebuah *junction table* relasi *many-to-many* antara tabel `players` dengan tabel `players` itu sendiri.

3.2.1.2.12 Tabel `player_building`

Tabel `player_building` berisi informasi mengenai *building* yang dimiliki oleh pemain. Rincian mengenai atribut-atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.20.

Tabel 3.20. Rincian atribut tabel `player_building`

Nama Atribut	Tipe Data	Keterangan
`BuildingID`	*Int*	*Primary key* tabel.
`PlayerID`	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain.
`Name`	*Varchar*(15)	Menunjukkan nama *building*.
`Level`	*Int*	Menunjukkan nilai *level building*.
`MaxXP`	*Int*	Menunjukkan nilai kapasitas maksimum XP *building*.
`CurrentXP`	*Int*	Menunjukkan nilai XP *building* saat ini.
`ProductionQuantity`	*Int*	Menunjukkan nilai produksi yang dapat dikeluarkan oleh *building*.
`ProductClaimed`	*Int*	Menunjukkan nilai produk yang telah diambil oleh pemain saat ini.

Nama Atribut	Tipe Data	Keterangan
PrefabName	*Varchar*(50)	Berisi informasi nama *prefab building* pada Unity.

Pada Tabel 3.20 dijelaskan mengenai keterangan masing-masing atribut pada tabel player_building beserta tipe datanya masing-masing. Tabel player_building mempunyai hubungan *one-to-many* dengan tabel players.

3.2.1.2.13 Tabel avatar_item

Tabel avatar_item berisi mengenai informasi *item-item* yang ada pada kategori *avatar* pada pemain. Rincian setiap atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.21.

Tabel 3.21. Rincian atribut tabel avatar_item

Nama Atribut	Tipe Data	Keterangan
ItemID	*Varchar*(10)	*Primary key* tabel.
Name	*Varchar*(50)	Menunjukkan nama *item*.
Slot	*Varchar*(10)	Menunjukkan *slot* tempat pemasangan *item* pada *avatar* pemain.
Gender	*Varchar*(10)	Menunjukkan jenis *gender item* pemain.

Pada Tabel 3.21 dijelaskan mengenai keterangan masing-masing atribut pada tabel `avatar_item` beserta tipe datanya masing-masing. Tabel *avatar*_`item` merupakan sebuah tabel *master*.

3.2.1.2.14 Tabel `player_avatar`

Tabel `player_avatar` berisi informasi mengenai *item-item* yang sedang dipakai saat ini pada *avatar* pemain. Rincian atribut-atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.22.

Tabel 3.22. Rincian atribut tabel `player_avatar`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Int*	*Primary key* tabel.
`PlayerID`	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain.
`ItemID`	*Varchar*(10)	*Foreign key* dari tabel *avatar_item*. Menunjukkan ID *avatar item*.

Pada Tabel 3.22 dijelaskan mengenai keterangan masing-masing atribut pada tabel `player_avatar` beserta tipe datanya masing-masing. Tabel `player_avatar` merupakan sebuah *junction table* relasi *many-to-many* antara tabel avatar_item dengan tabel `players`.

3.2.1.2.15 Tabel `gift`

Tabel `gift` berisi informasi mengenai *gift* yang di-*share* oleh pemain dan dapat diterima oleh beberapa pemain lain. Rincian atribut pada tabel ini dapat dilihat pada Tabel 3.23.

Tabel 3.23. Rincian atribut tabel `gift`

Nama Atribut	Tipe Data	Keterangan
`GiftID`	*Int*	*Primary key* tabel.
`GiftName`	*Varchar*(25)	Menunjukkan nama *gift*.
`Sender`	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain yang memberi *gift*.
`Quota`	*Int*	Menunjukkan nilai batasan *gift* yang dapat diambil oleh teman.
`Information`	*Varchar*(100)	Keterangan mengenai *gift* yang dikirim.

Pada Tabel 3.23 dijelaskan mengenai keterangan masing-masing atribut pada tabel `gift` beserta tipe datanya masing-masing. Tabel `gift` merupakan sebuah tabel *master*.

3.2.1.2.16 Tabel `gift_receiver`

Tabel `gift_receiver` menyimpan informasi mengenai pemain yang telah mengambil *gift* yang telah dikirim oleh pemain lain. Rincian mengenai atribut pada tabel ini dapat dilihat pada Tabel 3.24.

Tabel 3.24. Rincian atribut tabel `gift_receiver`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Int*	*Primary key* tabel.
`GiftID`	*Int*	*Foreign key* dari tabel *gift.* Menunjukkan ID *gift*.
`Code`	*Varchar*(25)	Berisi kode acak yang dapat ditukarkan pemain dengan suatu barang hadiah.
`Receiver`	*Varchar*(50)	*Foreign key* dari tabel *players.* Menunjukkan ID pemain yang mengambil *gift*.

Pada Tabel 3.24 dijelaskan mengenai keterangan masing-masing atribut pada tabel `gift_receiver` beserta tipe datanya masing-masing. Tabel `gift_receiver` merupakan sebuah *junction_table* penghubung relasi *many-to-many* antara tabel `players` dengan tabel `gift`.

3.2.1.2.17 Tabel `cards`

Tabel `cards` menyimpan informasi mengenai data kartu. Rincian setiap atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.25.

Tabel 3.25. Rincian atribut tabel `cards`

Nama Atribut	Tipe Data	Keterangan
`CardID`	*Varchar*(10)	*Primary key* tabel.
`Type`	*Varchar*(50)	Menunjukkan jenis kartu.
`Name`	*Varchar*(50)	Menunjukkan nama kartu.
`Element`	*Varchar*(25)	Menunjukkan elemen kartu.
`Point`	*Int*	Menunjukkan nilai *point* kartu.
`DeckPowerCost`	*Int*	Menunjukkan nilai *cost* kartu untuk dimasukkan dalam *deck*.
`SoulPowerCost`	*Int*	Menunjukkan nilai *cost* kartu untuk dikeluarkan pada saat *battle*.
`SellPrice`	*Int*	Nilai jual kartu apabila dijual di *shop*.
`Decription`	*Varchar*(100)	Deskripsi kartu.

Pada Tabel 3.25 dijelaskan mengenai keterangan masing-masing atribut pada tabel `cards` beserta tipe datanya masing-masing. Tabel `cards` merupakan sebuah tabel *master*.

3.2.1.2.18 Tabel `card_shop`

Tabel `card_shop` berisi mengenai informasi kartu yang ada pada *shop*. Rincian atribut-atribut pada tabel ini dapat dilihat pada Tabel 3.26.

Tabel 3.26. Rincian atribut tabel `card_shop`

Nama Atribut	Tipe Data	Keterangan
`TableID`	*Int*	*Primary key* tabel.
`CardID`	*Varchar*(10)	*Foreign key* dari tabel *cards*. Menunjukkan ID kartu.
`ShopPrice`	*Int*	Menunjukkan harga kartu.

Pada Tabel 3.26 dijelaskan mengenai keterangan masing-masing atribut pada tabel `card_shop` beserta tipe datanya masing-masing. Tabel `card_shop` mempunyai hubungan *one-to-many* - dengan tabel `cards`.

3.2.1.2.19 Tabel `player_card`

Tabel `player_card` menyimpan data kartu-kartu yang dimiliki oleh pemain. Rincian mengenai atribut-atribut dalam tabel ini dapat dilihat pada Tabel 3.27.

Tabel 3.27. Rincian atribut tabel `player_card`

Nama Atribut	Tipe Data	Keterangan
`DeckID`	*Int*	*Primary key* tabel.

PlayerID	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain.
CardID	*Varchar*(10)	*Foreign key* dari tabel *cards*. Menunjukkan ID kartu.
TotalQuantity	*Int*	Menunjukkan total kartu tertentu yang dimiliki pemain.
DeckQuantity	*Int*	Menunjukkan jumlah kartu tertentu pada *deck*.
TrunkQuantity	*Int*	Menunjukkan jumlah kartu tertentu pada *trunk*.

Pada Tabel 3.27 dijelaskan mengenai keterangan masing-masing atribut pada tabel player_card beserta tipe datanya masing-masing. Tabel player_card merupakan sebuah *junction table* relasi *many-to-many* antara tabel players dengan tabel cards.

3.2.1.2.20 Tabel trading_detail

Tabel trading_detail memberikan informasi mengenai *request* untuk bertukar kartu antar pemain. Rincian atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.28.

Tabel 3.28. Rincian atribut tabel trading_detail

Nama Atribut	Tipe Data	Keterangan
RequestID	*Int*	*Primary key* tabel.

SenderPlayer	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain pengirim *request*.
RequestedPlayer	*Varchar*(50)	*Foreign key* dari tabel *players*. Menunjukkan ID pemain penerima *request*.

PadaTabel 3.28 dijelaskan mengenai keterangan masing-masing atribut pada tabel trading_detail beserta tipe datanya masing-masing. Tabel trading_detail merupakan sebuah *junction table* relasi *many-to-many* antara tabel players dengan tabel players itu sendiri.

3.2.1.2.21 Tabel trade_request_detail

Tabel trade_request_detail berisi informasi mengenai data kartu-kartu yang ada pada *request* pertukaran kartu antar pemain. Rincian atribut yang ada pada tabel ini dapat dilihat pada Tabel 3.29.

Tabel 3.29. Rincian atribut tabel trade_request_detail

Nama Atribut	Tipe Data	Keterangan
TableID	*Int*	*Primary key* tabel.
RequestID	*Int*	*Foreign key* dari tabel *trading_detail.* Menunjukkan ID *trade request*.
CardID	*Varchar*(10)	*Foreign key* dari tabel

Nama Atribut	Tipe Data	Keterangan
		cards. Menunjukkan ID kartu.
`Quantity`	*Int*	Menunjukkan jumlah kartu tertentu yang ingin dipertukarkan.

Pada Tabel 3.29 dijelaskan mengenai keterangan masing-masing atribut pada tabel `trade_request_detail` beserta tipe datanya masing-masing. Tabel `trade_request_detail` mempunyai hubungan *one-to-many* dengan tabel `trading_detail`.

3.2.2. Perancangan *Stored Procedure*

Pada bagian ini diuraikan seluruh *stored procedure* yang ada dalam permainan *Card Warlock Saga* dalam bentuk tabel. Pada Tabel 3.30 berikut diuraikan mengenai perancangan *stored procedure* terkait dengan *web service* pada permainan *Card Warlock Saga.*

Tabel 3.30. Daftar *stored procedure*

Nama *Stored Procedure*	Jenis	Keterangan
`accept_friend_request`	*set*	Menerima permintaan pertemanan.
`clear_deck`	*set*	Mengosongkan *deck* kartu pemain.
`edit_avatar`	*set*	Mengubah *item avatar* yang dipakai pemain.

Nama *Stored Procedure*	Jenis	Keterangan
`find_friend`	*get*	Mencari data pemain lain.
`get_battle_list`	*get*	Mengambil data *list of battle* pada sebuah *dungeon*.
`get_friend_list`	*get*	Mengambil data *list of friend* dari seorang pemain.
`get_friend_request_list`	*get*	Mengambil data permintaan pertemanan dari pemain lain.
`get_list_avatar`	*get*	Mengambil data *avatar item* sesuai dengan data *gender* pemain.
`get_messages`	*get*	Mengambil data pesan masuk yang diterima oleh pemain dari pemain lain.
`get_monster_list`	*get*	Mengambil data *list of monster* pada satu buah *quest*.
`get_name_by_fb`	*get*	Mengambil data ID pemain berdasarkan Facebook ID yang telah terdaftar.
`get_partial_profile`	*get*	Mengambil data pemain secara umum, yakni hanya mencakup nama,

Nama *Stored Procedure*	Jenis	Keterangan
		job, *rank*, dan *level* pemain.
`get_player_avatar`	*get*	Mengambil data *avatar item* yang sedang dipakai oleh pemain saat ini.
`get_player_building`	*get*	Mengambil data *building* yang dimiliki oleh pemain.
`get_player_cards`	*get*	Mengambil data kartu yang dimiliki oleh pemain.
`get_player_deck`	*get*	Mengambil data kartu yang ada pada *deck* pemain.
`get_player_trunk`	*get*	Mengambil data kartu yang ada pada *trunk* pemain.
`get_profile`	*get*	Mengambil data seorang pemain secara keseluruhan.
`ignore_friend_request`	*set*	Mengabaikan permintaan pertemanan dari pemain lain.
`insert_to_deck`	*set*	Memasukkan kartu ke dalam *deck* pemain.
`receive_card`	*set*	Memasukkan data kartu

Nama *Stored Procedure*	Jenis	Keterangan
		baru ke dalam data kartu yang dimiliki oleh pemain.
`register`	*set*	Mendaftarkan pemain baru dalam permainan.
`remove_card`	*set*	Menghapus data kartu dari seorang pemain.
`remove_friend`	*set*	Menghapus pemain dari daftar teman.
`send_battle_result`	*set*	Mengirim hasil *battle* antar pemain.
`send_friend_request`	*set*	Mengirim permintaan pertemanan kepada pemain lain.
`send_message`	*set*	Mengirim pesan ke *inbox* pemain lain.
`show_battle_rank`	*get*	Mengambil data *battle* antara diri sendiri dengan pemain-pemain lain.
`show_player_ranking`	*get*	Mengambil data 5 pemain terbaik saat ini.
`update_building`	*set*	Meng-*update* data *building* pemain.
`update_player_data`	*set*	Meng-*update* data pemain setelah *quest* berakhir.

Nama *Stored Procedure*	Jenis	Keterangan
`login`	*set*	Melakukan proses *login* pemain ke dalam permainan.

Pada Tabel 3.30 diuraikan tentang seluruh *stored procedure* yang ada pada *database* permainan *Card Warlock Saga* beserta keterangan masing-masing *stored procedure* tersebut. Untuk rincian *parameter* setiap *stored procedure* dapat dilihat pada Tabel 3.31.

Tabel 3.31. Rincian *parameter stored procedure*

Nama *Stored Procedure*	*Parameter*
`accept_friend_request`	- Nama pemain (*string*). - Nama teman (*string*).
`clear_deck`	- Nama pemain (*string*).
`edit_avatar`	- Nama pemain (*string*). - Nama *avatar* kepala (*string*). - Nama *avatar* mata (*string*). - Nama *avatar* mulut (*string*).
`find_friend`	- Nama pemain (*string*).
`get_battle_list`	- Nama pemain (*string*). - ID *dungeon* (*string*).
`get_friend_list`	- Nama pemain (*string*).

Nama *Stored Procedure*	*Parameter*
`get_friend_request_list`	- Nama pemain (*string*).
`get_list_avatar`	- Nama pemain (*string*).
`get_messages`	- Nama pemain (*string*).
`get_monster_list`	- ID *quest* (*string*).
`get_name_by_fb`	- ID Facebook pemain (*string*).
`get_partial_profile`	- Nama pemain (*string*).
`get_player_avatar`	- Nama pemain (*string*).
`get_player_building`	- Nama pemain (*string*).
`get_player_cards`	- Nama pemain (*string*).
`get_player_deck`	- Nama pemain (*string*).
`get_player_trunk`	- Nama pemain (*string*).
`get_profile`	- Nama pemain (*string*).
`ignore_friend_request`	- Nama pemain (*string*). - Nama teman (*string*).
`insert_to_deck`	- Nama pemain (*string*). - Nama kartu (*string*). - Jumlah kartu (*int*).
`receive_card`	- Nama pemain (*string*). - Nama kartu (*string*). - Jumlah kartu (*int*).

Nama *Stored Procedure*	*Parameter*
`Register`	- *Username* pemain (*string*). - *Password* pemain (*string*). Adapun *input* yang bersifat *optional* yakni sebagai berikut. - Alamat *e-mail* pemain (*string*). - ID Facebook pemain (*string*).
`remove_card`	- Nama pemain (*string*). - Nama kartu (*string*). - Jumlah kartu (*int*).
`remove_friend`	- Nama pemain (*string*). - Nama teman (*string*).
`send_battle_result`	- Nama pemain (*string*). - Nama teman (*string*).
`send_friend_request`	- Nama pemain (*string*). - Nama teman (*string*).
`send_message`	- Nama pemain (*string*). - Nama teman (*string*). - Subjek pesan (*string*). - Isi pesan (*string*).
`show_battle_rank`	- Nama pemain (*string*).
`show_player_ranking`	-

Nama *Stored Procedure*	*Parameter*
`update_building`	- Nama pemain (*string*). - Nama *building* (*string*). - Poin XP dan maksimal XP *building* (*int*).
`update_player_data`	- Nama pemain (*string*). - Data *rank* pemain(*string*). - Poin *gold*, XP, dan *level* pemain (*int*).
`Login`	- Nama pemain (*string*). - *Password* pemain (*string*).

Setiap *stored procedure* mempunyai jenis fungsi masing-masing, bisa *set* ataupun *get*. Untuk fungsi *set*, *stored procedure* melakukan aksi berupa *insert*, *update* ataupun *delete* data. Kembalian dari fungsi *set* adalah berupa keterangan berhasil tidaknya proses *insert*, *update* ataupun *delete* data yang disimpan pada *query result* dalam bentuk *string*. Sedangkan untuk fungsi *get*, *stored procedure* melakukan aksi *select* data yang dibutuhkan. Fungsi *get* mengembalikan 2 macam *return value*, yakni *query result* dan hasil *select* data. *Query result* dapat mempunyai 3 kemungkinan nilai *query result*, yakni sebagai berikut.

a. Nilai positif (lebih dari 0), yang berarti *input* data benar, serta menghasilkan satu atau lebih data hasil *select* .
b. Nilai 0. Pada fungsi *get* hal ini berarti *input* data benar, namun data yang di-*select* kosong, sedangkan pada fungsi *set* hal ini berarti terjadi kesalahan *input* / terjadi *exception*.

c. Nilai negatif (kurang dari 0), yang berarti *input* data tidak *valid* / terjadi *exception* seperti adanya *error* pada *server*, atau adanya *typo* pada penulisan *code* pada *server*.

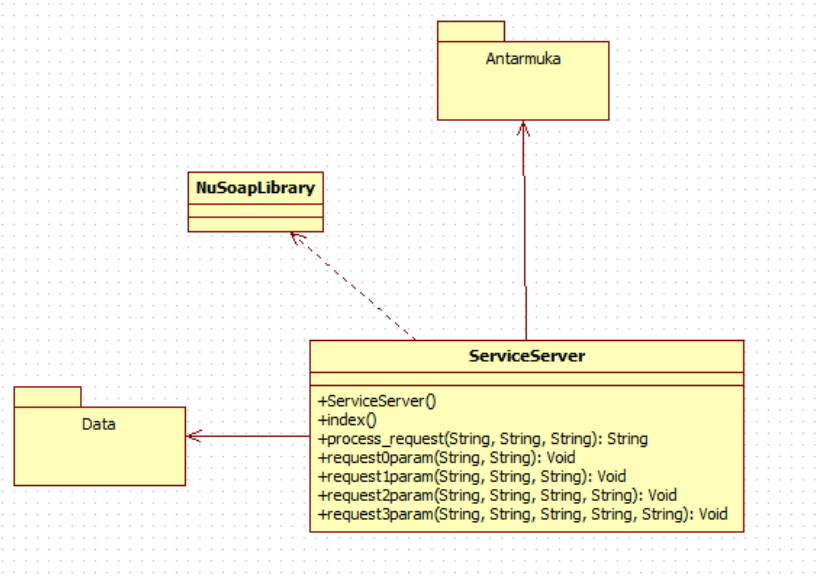
3.2.3. Perancangan Diagram Kelas

Perancangan diagram kelas berisi rancangan dari kelas-kelas yang digunakan untuk membangun sistem. Terdapat 3 jenis kelas pada perancangan ini, yakni: kelas *model* (data), antarmuka (*view)* dan kelas *controller* (kontrol). Berikut dijelaskan mengenai perancangan diagram kelas untuk kelas data dan kontrol.

3.2.3.1. Diagram Kelas Lapisan Kontrol

Pada lapisan kontrol terdapat 2 buah kelas yang saling berhubungan, yaitu *NuSOAPLibrary* dan *ServiceServer*.

Kelas *NuSOAPLibrary* merupakan sebuah kelas *library* yang digunakan untuk membangun SOAP *web service*. Kelas *ServiceServer* merupakan kelas yang mengimplementasikan *library* pada kelas *NuSOAPLibrary* untuk membangun SOAP *web service* yang dapat menyediakan fungsi-fungsi yang dibutuhkan dalam permainan *Card Warlock Saga*. Kelas *ServiceServer* kemudian akan mengembalikan nilai hasil eksekusi fungsi ke kelas *view*. Data yang ada pada kelas *view* inilah yang diterima oleh pengguna.



Gambar 3.1. Diagram Kelas Lapisan Kontrol

Pada Gambar 3.1 ditunjukkan diagram kelas lapisan kontrol yang menunjukkan hubungan antara ketiga kelas pada lapisan *control* dengan lapisan *view* dan *model*.

3.2.3.2. Diagram Kelas Lapisan Data

Pada lapisan data terdapat beberapa kelas yaitu: *Avatar*, *Player*, *MonsterQuest*, *Request*, *Card*, *Message*, *Battle* dan *Building*. Kelas lapisan data hanya dapat diakses melalui kelas kontrol, di mana kelas kontrol mengakses fungsi-fungsi yang ada pada kelas lapisan data sesuai dengan data apa yang dibutuhkan.

Gambar 9.6 menunjukkan diagram kelas lapisan data. Masing-masing kelas hanya dapat diakses melalui kelas kontrol. Setiap kelas pada lapisan data mempunyai fungsi masing-masing yang berhubungan dengan data atau *model* pada *database*.

BAB IV
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bab ini berisi proses implementasi dari setiap kelas pada semua modul. Bahasa pemrograman yang digunakan adalah bahasa pemrograman PHP pada sisi *web service* dan bahasa SQL pada sisi basis data.

4.1. Implementasi Basis Data

Implementasi basis data dibagi menjadi 2 tahapan, yakni implementasi tabel dan *stored procedure*.

4.1.1. Implementasi Tabel

Proses implementasi tabel pada *database* dimulai dengan perancangan tabel dan atributnya, pembuatan tabel dan penyambungan relasi antar tabel.

4.1.1.1. Tabel `players`

Tabel ini merupakan sebuah tabel *master* yang digunakan untuk menyimpan data-data pemain. Tabel *players* mempunyai beberapa relasi dengan tabel-tabel lain, di antaranya adalah sebagai berikut.

a. Relasi *many-to-many* dengan tabel `cards`, yang menghasilkan tabel `player_card` sebagai *junction table* yang menyimpan data kartu yang dimiliki oleh setiap pemain.
b. Relasi *many-to-many* dengan tabel `players`, yang menghasilkan beberapa *junction table*, yakni sebagai berikut.
    - Tabel `battle_detail` sebagai tabel yang menyimpan data *battle* antar pemain.

- Tabel `trading_detail` sebagai tabel yang menyimpan data *trading request* antar pemain.
- Tabel `player_friend` sebagai tabel yang menyimpan data pertemanan antar pemain.
- Tabel `player_request` sebagai tabel yang menyimpan data *request* berupa energi atau pertemanan antar pemain.

c. Relasi *one-to-many* dengan tabel `player_message` sebagai tabel yang menyimpan data pertukaran pesan antar pemain.
d. Relasi *one-to-many* dengan tabel `player_building` yang menyimpan data *building* yang dimiliki oleh setiap pemain.
e. Relasi *many-to-many* dengan tabel `avatar_item`, yang menghasilkan tabel `player_avatar` sebagai *junction table* yang menyimpan data *avatar* yang dimiliki oleh setiap pemain.
f. Relasi *one-to-many* dengan tabel `trade_request_detail` yang menyimpan data *request* kartu yang akan dipertukarkan dengan pemain lain.
g. Relasi *one-to-many* dengan tabel `gift` yang menyimpan data *gift* yang dikirim oleh setiap pemain.
h. Relasi *many-to-many* dengan tabel `gift`, yang menghasilkan tabel `gift_receiver` sebagai *junction table* yang menyimpan data pemain yang mengklaim *gift* yang telah dikirim oleh pemain lain.
i. Relasi *one-to-many* dengan tabel `setting` yang menyimpan data konfigurasi pembatasan aksi pemain dalam permainan.
j. Relasi *many-to-many* dengan tabel `quest_battle`, yang menghasilkan tabel `player_quest` sebagai *junction table* yang menyimpan data *quest* yang dimiliki oleh setiap pemain.

4.1.1.2. Tabel `dungeon`

Tabel ini merupakan sebuah tabel *master* yang digunakan untuk menyimpan data *dungeon* yang ada pada permainan. Tabel `dungeon` mempunyai relasi *one-to-many* dengan tabel `quest_battle` yang menyimpan data *battle* yang dimiliki oleh setiap *dungeon*.

4.1.1.3. Tabel `monster`

Tabel ini merupakan sebuah tabel *master* yang digunakan untuk menyimpan data *monster* yang ada pada permainan. Tabel `monster` mempunyai relasi *many-to-many* dengan tabel `quest_battle` yang menghasilkan tabel `battle_monster` sebagai *junction table*-nya yang menyimpan data *monster* yang ada pada setiap *battle* pada *dungeon*.

4.1.1.4. Tabel `cards`

Tabel ini merupakan sebuah tabel *master* yang digunakan untuk menyimpan data kartu yang ada pada permainan. Tabel `cards` mempunyai relasi-relasi dengan tabel lain, yakni sebagai berikut.

a. Relasi *one-to-many* dengan tabel `card_shop` yang menyimpan data kartu yang dijual di *shop*.
b. Relasi *one-to-many* dengan tabel `trade_request_detail` yang menyimpan data *request* kartu yang akan dipertukarkan dengan pemain lain.
c. Relasi *many-to-many* dengan tabel `players` yang menghasilkan tabel `player_card` sebagai *junction table* yang menyimpan data kartu yang dimiliki setiap pemain.

4.1.1.5. Tabel `player_building`

Tabel ini merupakan tabel yang mempunyai relasi *one-to-many* dengan tabel `players` sebagai tabel yang menyimpan data *building* pemain.

4.1.1.6. Tabel `player_avatar`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `players` dengan tabel `avatar_item` sebagai tabel yang menyimpan data *avatar* yang dipakai setiap pemain.

4.1.1.7. Tabel `battle_detail`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `players` dengan dirinya sendiri sebagai tabel yang menyimpan data *battle* antar pemain.

4.1.1.8. Tabel `avatar_item`

Tabel ini menyimpan data seluruh *avatar* yang ada pada permainan. Tabel ini mempunyai relasi *many-to-many* dengan tabel `players` dan menghasilkan tabel `avatar_item` sebagai *junction table* yang menyimpan data *avatar* yang dipakai setiap pemain.

4.1.1.9. Tabel `player_card`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `players` dengan tabel `cards` sebagai tabel yang menyimpan data kartu yang dimiliki setiap pemain.

4.1.1.10. Tabel `card_shop`

Tabel ini mempunyai relasi *one-to-many* dengan tabel `cards` sebagai tabel yang menyimpan data kartu yang dijual pada *shop*.

4.1.1.11. Tabel `trading_detail`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `players` dengan dirinya sendiri sebagai tabel yang menyimpan data *trading request* antar pemain. Selain itu, tabel ini juga mempunyai relasi *one-to-many* dengan tabel `trade_request_detail`.

4.1.1.12. Tabel `trade_request_detail`

Tabel ini mempunyai relasi *one-to-many* dengan tabel `players`, tabel `cards` dan tabel `trading_detail` sebagai tabel yang menyimpan data kartu yang di-*request* untuk dipertukarkan antar pemain.

4.1.1.13. Tabel `quest_battle`

Tabel ini mempunyai beberapa relasi dengan tabel-tabel lain, yakni sebagai berikut.

a. Relasi *many-to-many* dengan tabel `players` sebagai tabel yang menyimpan data *quest* pemain.
b. Relasi *one-to-many* dengan tabel `dungeon` sebagai tabel yang menyimpan data *quest* atau *battle* yang ada pada setiap *dungeon*.
c. Relasi *many-to-many* dengan tabel `monster` yang menghasilkan tabel *battle_monster* sebagai tabel yang menyimpan data *monster* yang ada pada setiap *quest* atau *battle*.

4.1.1.14. Tabel `player_quest`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `players` dengan tabel `quest_battle` sebagai tabel yang menyimpan data *quest* pemain.

4.1.1.15. Tabel `battle_monster`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `quest_battle` dengan tabel `monster` sebagai tabel yang menyimpan data *monster* tiap *quest* atau *battle*.

4.1.1.16. Tabel `player_request`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `players` dengan dirinya sendiri sebagai tabel yang menyimpan data *request* energi dan pertemanan antar pemain.

4.1.1.17. Tabel `player_friend`

Tabel ini merupakan *junction table* relasi *many-to-many* antara tabel `players` dengan dirinya sendiri sebagai tabel yang menyimpan data hubungan pertemanan antar pemain.

4.1.1.18. Tabel `player_message`

Tabel ini mempunyai relasi *one-to-many* dengan tabel `players` sebagai tabel yang menyimpan data pertukaran pesan antar pemain.

4.1.1.19. Tabel `gift`

Tabel ini mempunyai relasi *one-to-many* dengan tabel `gift_receiver` sebagai tabel yang menyimpan data pemain yang mengklaim *gift* yang ada, dan relasi *one-to-many* dengan tabel `players` sebagai tabel yang menyimpan data pemain yang mengirimkan *gift*.

4.1.1.20. Tabel `gift`

Tabel ini mempunyai relasi *one-to-many* dengan tabel `gift_receiver` sebagai tabel yang menyimpan data pemain yang mengklaim *gift* yang ada, dan relasi *one-to-many* dengan

tabel `players` sebagai tabel yang menyimpan data pemain yang mengirimkan *gift*.

4.1.1.21. Tabel `setting`

Tabel ini mempunyai relasi *one-to-many* dengan tabel `players` sebagai tabel yang menyimpan data konfigurasi pembatasan aksi pemain pada permainan.

4.1.2. Implementasi *Stored Procedure*

Pada sub-bab ini dijabarkan satu per satu implementasi *stored procedure* yang ada pada *database*.

4.1.2.1. Fungsi `accept_energy_request`

Fungsi ini mengimplementasikan proses penerimaan *request* energi dari pemain lain. Terdapat 2 nilai masukan pada fungsi ini yang sama-sama berupa ID pemain dengan nama *player_one* dan *player_two*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.1.

```
DECLARE energy_left INT;
DECLARE request_id INT;
DECLARE is_valid VARCHAR(1);
SELECT 1 INTO is_valid FROM player_request
    WHERE     RequestedPlayer=player_one     AND
SenderPlayer=player_two
    AND Information='Energy Sent'
    LIMIT 1;
SELECT  Energy  INTO  energy_left  FROM  players
WHERE PlayerID=player_one;

IF(is_valid!="") THEN
    IF(energy_left>=25) THEN
        SELECT   0   AS   result,   CONCAT("Your
current  energy  is  full,  you  can  accept  this
extra energy later") AS info;
```

```
    ELSE
        UPDATE players SET Energy=energy_left+1
WHERE PlayerID=player_one;
        SELECT requestid INTO request_id FROM
player_request
            WHERE     RequestedPlayer=player_one
AND SenderPlayer=player_two
            AND Information='Energy Sent'
            ORDER BY RequestTime ASC LIMIT 1;
        DELETE    FROM    player_request    WHERE
RequestID=request_id;
        SELECT   1   AS   result,   CONCAT("Extra
energy received") AS info;
    END IF;
ELSE
    SELECT  -1  AS  result,  CONCAT("No  Request
Found") AS info;
END IF;
```

Kode Sumber 4.1. Potongan fungsi `accept_energy_request`

4.1.2.2. Fungsi `accept_friend_request`

Fungsi ini mengimplementasikan proses penerimaan *request* pertemanan dari pemain lain. Terdapat 2 nilai masukan pada fungsi ini yang sama-sama berupa ID pemain dengan nama *player_one* dan *player_two*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.2

```
DECLARE is_valid VARCHAR(1);
SELECT 1 INTO is_valid FROM player_request
WHERE        SenderPlayer=player_two         AND
RequestedPlayer=player_one                   AND
Information='friend request sent';

IF(is_valid!="") THEN
    INSERT          INTO          player_friend
VALUES('',player_one, player_two);
```

```
    INSERT          INTO          player_friend
VALUES('',player_two, player_one);
    DELETE FROM player_request
    WHERE       SenderPlayer=player_two       AND
RequestedPlayer=player_one                    AND
Information='friend request sent';
    SELECT  1  AS  result,  CONCAT("You  and  ",
player_two, " are now friends") AS info;
ELSE
    SELECT  0  AS  result,  CONCAT("No  Friend
Request Found") AS info;
END IF;
```

Kode Sumber 4.2. Potongan fungsi `accept_friend_request`

4.1.2.3. Fungsi `clear_deck`

Fungsi ini mengimplementasikan proses pengosongan *deck* pemain. Fungsi ini mempunyai satu nilai masukan yakni ID pemain dengan nama *player_id*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.3.

```
DECLARE dp_point INT;
SELECT MaxDP INTO dp_point FROM players WHERE
PlayerID=player_id;
UPDATE player_card SET DeckQuantity=0,
TrunkQuantity=TotalQuantity WHERE
PlayerID=player_id;

SELECT 1 AS result, CONCAT("Deck is Cleared")
AS info;
```

Kode Sumber 4.3. Potongan fungsi `clear_deck`

4.1.2.4. Fungsi `decline_trade_request`

Fungsi ini mengimplementasikan proses penolakan *request* pertukaran kartu antar pemain. Fungsi ini mempunyai satu nilai masukan yakni ID *request* dengan nama *req_id*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.4.

```
DECLARE is_valid INT;
SELECT 1 INTO is_valid FROM trading_detail
WHERE RequestID=req_id LIMIT 1;
IF(is_valid!="") THEN

    DELETE FROM trading_detail WHERE
RequestID=req_id;

    SELECT 1 AS result, CONCAT("Trade Request
Declined") AS info;
ELSE

    SELECT 0 AS result, CONCAT("Invalid ID") AS
info;
END IF;
```

Kode Sumber 4.4. Potongan fungsi `decline_trade_request`

4.1.2.5. Fungsi `edit_avatar`

Fungsi ini mengimplementasikan proses perubahan *avatar* pemain saat ini dan menyimpan perubahan tersebut. Fungsi ini mempunyai 4 buah nilai masukan yakni : ID pemain dengan nama *player_id*, ID *avatar* untuk bagian kepala dengan nama *head_slot*, ID *avatar* untuk bagian mata dengan nama *eye_slot* dan ID *avatar* untuk bagian mulut dengan nama *mouth_slot*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.5.

```
DECLARE item_id VARCHAR(10);
DELETE FROM player_avatar WHERE
PlayerID=player_id;

SELECT av.ItemID INTO item_id FROM avatar_item
av WHERE av.Name=head_slot;
INSERT INTO player_avatar VALUES('',player_id,
item_id);

SELECT av.ItemID INTO item_id FROM avatar_item
av WHERE av.Name=eye_slot;
INSERT INTO player_avatar VALUES('',player_id,
item_id);

SELECT av.ItemID INTO item_id FROM avatar_item
av WHERE av.Name=mouth_slot;
INSERT INTO player_avatar VALUES('',player_id,
item_id);

SELECT 1 AS result, CONCAT("Avatar has been
updated") AS info;
```

Kode Sumber 4.5. Potongan fungsi `edit_avatar`

4.1.2.6. Fungsi `find_friend`

Fungsi ini mengimplementasikan proses pencarian data pemain lain dengan nama tertentu. Terdapat 2 nilai masukan pada fungsi ini yang sama-sama berupa ID pemain dengan nama *player_one* dan *player_two*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.6.

```
DECLARE is_on_list VARCHAR(1);
IF(friend_name!="") THEN
    SELECT 1 INTO is_on_list FROM player_friend
WHERE PlayerOne=player_id AND
PlayerTwo=friend_name;
    IF(is_on_list!="") THEN
```

```
        SELECT PlayerTwo AS friend FROM
player_friend WHERE PlayerOne=player_id AND
PlayerTwo=friend_name;
    ELSE
        SELECT 0, CONCAT("Data could not be
found");
    END IF;
END IF;
```

Kode Sumber 4.6. Potongan fungsi `find_friend`

4.1.2.7. Fungsi `get_battle_list`

Fungsi ini mengimplementasikan proses pencarian data *battle list* yang ada pada suatu *dungeon*. Fungsi ini mempunyai satu buah nilai masukan berupa ID *dungeon* dengan nama *dungeon_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.6.

4.1.2.8. Fungsi `get_card_request_list`

Fungsi ini mengimplementasikan proses pencarian data *card list* yang ada pada suatu *trading request*. Fungsi ini mempunyai satu buah nilai masukan berupa ID *request* dengan nama *req_id*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.7.

```
DECLARE is_valid INT;
SELECT 1 INTO is_valid FROM
trade_request_detail WHERE RequestID=req_id
LIMIT 1;
IF(is_valid!="") THEN
    SELECT 1 AS result, t.PlayerID, c.Name,
t.Quantity
    FROM trade_request_detail t INNER JOIN
cards c ON t.CardID=c.CardID
    WHERE t.RequestID=req_id ORDER BY t.TableID
ASC;
ELSE
```

```
    SELECT 0 AS result, CONCAT("Invalid ID") AS
info;
END IF;
```

Kode Sumber 4.7. Potongan fungsi `get_card_request_list`

4.1.2.9. Fungsi `get_energy_request_list`

Fungsi ini mengimplementasikan proses pencarian data *request* energi yang dikirim kepada seorang pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.8.

```
DECLARE is_valid VARCHAR(1);
SELECT 1 INTO is_valid FROM player_request
WHERE RequestedPlayer=player_id AND
Information='Energy Sent' LIMIT 1;

IF(is_valid!="") THEN
    SELECT 1 AS result, SenderPlayer AS player
FROM player_request WHERE
RequestedPlayer=player_id AND
Information='Energy Sent';
ELSE
    SELECT 0 AS result, CONCAT("Data Not
Found") AS info;
END IF;
```

Kode Sumber 4.8. Potongan fungsi `get_energy_request_list`

4.1.2.10. Fungsi `get_friend_list`

Fungsi ini mengimplementasikan proses pencarian data *friend list* dari seorang pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.9.

4.1.2.11. Fungsi `get_friend_request_list`

Fungsi ini mengimplementasikan proses pencarian data *friend request list* dari seorang pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id.* Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.9.

```
DECLARE is_valid VARCHAR(1);

SELECT 1 INTO is_valid FROM player_request
WHERE RequestedPlayer=player_id AND
Information='Friend Request Sent' LIMIT 1;
IF(is_valid!="") THEN

    SELECT 1 AS result, p.PlayerID, p.Job,
p.Rank, p.Level
    FROM player_request r INNER JOIN players p
ON p.PlayerID=r.SenderPlayer
    WHERE RequestedPlayer=player_id AND
Information='friend request sent';
ELSE

    SELECT 0 AS result, CONCAT("Data Not
Found") AS info;
END IF;
```

Kode Sumber 4.9. Potongan fungsi `get_friend_request_list`

4.1.2.12. Fungsi `get_list_avatar`

Fungsi ini mengimplementasikan proses pengambilan data *avatar list* yang *available* pada seorang pemain menyesuaikan *gender* pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id.* Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.10.

```
DECLARE gender VARCHAR(10);
```

```
SELECT p.Gender INTO gender FROM players p
WHERE PlayerID=player_id;

SELECT av.Name, av.Slot FROM avatar_item av
WHERE av.Gender=gender;
```

Kode Sumber 4.10. Potongan fungsi `get_list_avatar`

4.1.2.13. Fungsi `get_messages`

Fungsi ini mengimplementasikan proses pengambilan data pesan masuk yang diterima oleh pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.12.

4.1.2.14. Fungsi `get_monster_list`

Fungsi ini mengimplementasikan proses pengambilan data *monster list* dalam sebuah *dungeon*. Fungsi ini mempunyai satu buah nilai masukan berupa ID *battle* dengan nama *battle_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.13.

4.1.2.15. Fungsi `get_name_by_fb`

Fungsi ini mengimplementasikan proses pengambilan data nama pemain berdasarkan Facebook ID yang dimilikinya. Fungsi ini mempunyai satu buah nilai masukan berupa Facebook ID pemain dengan nama *player_fb_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.14.

4.1.2.16. Fungsi `get_partial_profile`

Fungsi ini mengimplementasikan proses pencarian data pemain secara umum berupa keterangan nama, *job*, *rank* dan *level* pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.11.

```
DECLARE is_valid INT;
SELECT 1 INTO is_valid FROM players WHERE
PlayerID=player_id;
IF(is_valid!="") THEN
    SELECT 1 AS result, p.PlayerID, p.Job,
p.Rank, p.Level
    FROM players p WHERE p.PlayerID=player_id;
ELSE
    SELECT 0 AS result, CONCAT("Data Could Not
Be Found") AS info;
END IF;
```

Kode Sumber 4.11. Potongan fungsi `get_partial_profile`

4.1.2.17. Fungsi `get_player_avatar`

Fungsi ini mengimplementasikan proses pengambilan data *avatar* pemain saat ini. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.16.

4.1.2.18. Fungsi `get_player_building`

Fungsi ini mengimplementasikan proses pengambilan data *building* yang dimiliki oleh pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.17.

4.1.2.19. Fungsi `get_player_deck`

Fungsi ini mengimplementasikan proses pengambilan data kartu pemain yang ada pada *deck*. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.19.

4.1.2.20. Fungsi `get_player_trunk`

Fungsi ini mengimplementasikan proses pengambilan data kartu pemain yang ada pada *trunk*. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.21.

4.1.2.21. Fungsi `get_profile`

Fungsi ini mengimplementasikan proses pengambilan data keseluruhan dari pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.22.

4.1.2.22. Fungsi `get_trade_request_list`

Fungsi ini mengimplementasikan proses pengambilan data *trade request* yang diterima oleh pemain. Fungsi ini mempunyai satu buah nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.23.

4.1.2.23. Fungsi `ignore_friend_request`

Fungsi ini mengimplementasikan proses penolakan *request* pertemanan yang dikirim kepada pemain. Fungsi ini mempunyai 2 buah nilai masukan yang sama sama berupa ID pemain dengan nama *player_one* dan *player_two*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.24.

4.1.2.24. Fungsi `insert_card_request`

Fungsi ini mengimplementasikan proses pemasukan data kartu ke dalam *request* pertukaran kartu yang dikirim kepada pemain lain. Fungsi ini mempunyai 4 buah nilai masukan, yaitu: ID pemain dengan nama *player_id*, ID *request* dengan nama *request_id*, nama kartu dengan nama *card_name* dan jumlah kartu dengan

nama *quantity*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.12.

```
DECLARE card_id VARCHAR(50);
SELECT c.CardID INTO card_id FROM cards c
WHERE c.Name=card_name;
IF(card_id!="") THEN
    INSERT INTO trade_request_detail
VALUES('',player_id, request_id, card_id,
card_quantity);
    SELECT 1 AS result, CONCAT("Trade Request
Sent") AS info;
ELSE
    SELECT 0 AS result, CONCAT("Invalid card
name") AS info;
END IF;
```

Kode Sumber 4.12. Potongan fungsi `insert_card_request`

4.1.2.25. Fungsi `insert_to_deck`

Fungsi ini mengimplementasikan proses pemasukan data kartu ke dalam *deck* kartu pemain. Fungsi ini mempunyai 3 buah nilai masukan, yaitu: ID pemain dengan nama *player_id*, nama kartu dengan nama *card_name* dan jumlah kartu dengan nama *quantity*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.26.

4.1.2.26. Fungsi `login`

Fungsi ini mengimplementasikan proses *login* pemain ke dalam permainan. Fungsi ini mempunyai 2 buah nilai masukan, yaitu: ID pemain dengan nama *p_username*, dan pemain dengan nama *p_password*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.13.

```
DECLARE login_status VARCHAR(1);
SELECT 1 INTO login_status FROM players p WHERE
p.PlayerID=p_username AND p.Password=p_pass;
```

```
IF(login_status != "") THEN
    SELECT 1 AS result, CONCAT("Login
Succesful") AS info;
ELSE
    SELECT 0 AS result, CONCAT("Login Failed.
Incorrect Username or Password") AS info;
END IF;
```

Kode Sumber 4.13. Potongan fungsi `login`

4.1.2.27. Fungsi `receive_card`

Fungsi ini mengimplementasikan proses penerimaan kartu kepada pemain. Fungsi ini mempunyai 3 buah nilai masukan, yaitu: ID pemain dengan nama *player_id*, nama kartu dengan nama *card_name* dan jumlah kartu dengan nama *quantity*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.14.

```
DECLARE is_on_deck VARCHAR(1);
DECLARE card_id VARCHAR(10);
SELECT c.CardID INTO card_id FROM cards c WHERE
c.Name = card_name LIMIT 1;
IF(quantity<=0) THEN
    SELECT -1 AS result, CONCAT("Invalid Card
Quantity");
ELSE
    IF(card_id!="") THEN
        SELECT 1 INTO is_on_deck FROM
player_card WHERE PlayerID=player_id AND
CardID=card_id;
        IF(is_on_deck!="") THEN
            UPDATE player_card SET
TotalQuantity=TotalQuantity+quantity,
TrunkQuantity=TrunkQuantity+quantity
            WHERE PlayerID=player_id AND
CardID=card_id;
        ELSE
            INSERT INTO player_card VALUES('',
```

```
player_id, card_id, quantity, 0, quantity);
        END IF;
        SELECT 1 AS result, CONCAT("Item
Received") AS info;
    ELSE
        SELECT 0 AS result, CONCAT("Invalid
Card Name") AS info;
    END IF;
END IF;
```

Kode Sumber 4.14. Potongan fungsi `receive_card`

4.1.2.28. Fungsi `register`

Fungsi ini mengimplementasikan proses pendaftaran pemain baru ke dalam permainan. Fungsi ini mempunyai 5 buah nilai masukan, yaitu: ID pemain dengan nama *p_username*, Facebook ID pemain dengan nama *p_fb*, alamat *e-mail* pemain dengan nama *p_email*, *job* pilihan pemain dengan nama *p_job* dan *password* pemain dengan nama *p_pass*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.15.

```
DECLARE username_isvalid VARCHAR(1);
DECLARE is_registered VARCHAR(1);
SELECT 1 INTO username_isvalid FROM players p
WHERE p.PlayerID=p_username;
IF(p_fb!="") THEN
    SELECT 1 INTO is_registered FROM players p
WHERE p.FB_ID=p_fb LIMIT 1;
END IF;

IF(is_registered!="") THEN
    SELECT -1 AS result, CONCAT("Registration
Failed. Facebook is already registered") AS
info;
ELSE
    IF(username_isvalid!="") THEN
        SELECT 0 AS result, CONCAT("Username is
not valid or has been taken") AS info;
```

```
    ELSE
        INSERT INTO players VALUES(p_username,
p_fb,              p_pass,              p_email,
100,100,40,3,0,0,0,10,p_job,'D',1,'Female');
        SELECT       1       AS       result,
CONCAT("Registration Success") AS info;
    END IF;
END IF;
```

Kode Sumber 4.15. Potongan fungsi `register`

4.1.2.29. Fungsi *`remove_card`*

Fungsi ini mengimplementasikan proses penghapusan kartu dari pemain. Fungsi ini mempunyai 3 buah nilai masukan, yaitu: ID pemain dengan nama *player_id*, nama kartu dengan nama *card_name* dan jumlah kartu dengan nama *quantity*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.30.

4.1.2.30. Fungsi `remove_friend`

Fungsi ini mengimplementasikan proses penghapusan pemain lain dari daftar teman pemain. Terdapat 2 nilai masukan pada fungsi ini yang sama-sama berupa ID pemain dengan nama *player* dan *friend_name*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.31.

4.1.2.31. Fungsi `send_battle_result`

Fungsi ini mengimplementasikan proses pengiriman data hasil *battle* antar pemain. Terdapat 2 nilai masukan pada fungsi ini yang sama-sama berupa ID pemain dengan nama *player_one* dan *player_two*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.16.

```
DECLARE had_battle INT;
DECLARE win_points INT;
DECLARE lost_points INT;
```

```
SELECT Battle_Won INTO win_points FROM players
WHERE PlayerID=player_win;
SELECT Battle_Lost INTO lost_points FROM
players WHERE PlayerID=player_lost;

UPDATE players SET Battle_Won=win_points+1
WHERE PlayerID=player_win;
UPDATE players SET Battle_Lost=lost_points+1
WHERE PlayerID=player_lost;
SELECT 1 INTO had_battle FROM battle_detail
WHERE PlayerOne=player_win AND
PlayerTwo=player_lost;
IF(had_battle!="") THEN
    SELECT Win INTO win_points FROM
battle_detail WHERE PlayerOne=player_win AND
PlayerTwo=player_lost;
    SELECT Lose INTO lost_points FROM
battle_detail WHERE PlayerOne=player_lost AND
PlayerTwo=player_win;
    UPDATE battle_detail SET Win=win_points+1
WHERE PlayerOne=player_win AND
PlayerTwo=player_lost;
    UPDATE battle_detail SET Lose=lost_points+1
WHERE PlayerOne=player_lost AND
PlayerTwo=player_win;
ELSE
    INSERT INTO battle_detail
VALUES('',player_win, player_lost,1,0);
    INSERT INTO battle_detail
VALUES('',player_lost, player_win,0,1);
END IF;
SELECT 1 as result, CONCAT("Battle Result
Updated") as info;
```

Kode Sumber 4.16. Potongan fungsi `send_battle_result`

4.1.2.32. Fungsi `send_energy_request`

Fungsi ini mengimplementasikan proses pengiriman *request* energi kepada pemain lain. Terdapat 2 nilai masukan pada fungsi

ini yang sama-sama berupa ID pemain dengan nama *sender_player* dan *requested_player*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.17.

```
DECLARE start_time DATETIME;
DECLARE end_time DATETIME;
DECLARE time_dif INT;
DECLARE is_valid VARCHAR(1);

SELECT RequestTime INTO start_time FROM
player_request
WHERE SenderPlayer=sender_player AND
RequestedPlayer=requested_player
AND Information LIKE '%nergy%'
ORDER BY RequestTime DESC LIMIT 1;

SELECT NOW() INTO end_time;
SELECT DATEDIFF(end_time, start_time) INTO
time_dif;

SELECT 1 INTO is_valid FROM players WHERE
PlayerID=requested_player;

IF(is_valid!="") THEN
    IF(time_dif<1) THEN

        SELECT -1 AS result, CONCAT("You can
only send extra energy once per day for each
friend") AS info;
    ELSE
        INSERT INTO player_request
VALUES('',sender_player,requested_player,NOW(),
'Energy Sent');
        SELECT 1 AS result, CONCAT("Energy
Sent") AS info;
    END IF;
ELSE
    SELECT 0 AS result, CONCAT("Invalid Player
```

```
Name") AS info;
END IF;
```

Kode Sumber 4.17. Potongan fungsi `send_energy_request`

4.1.2.33. Fungsi `send_friend_request`

Fungsi ini mengimplementasikan proses pengiriman *request* pertemanan kepada pemain lain. Terdapat 2 nilai masukan pada fungsi ini yang sama-sama berupa ID pemain dengan nama *sender_player* dan *requested_player*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.34.

4.1.2.34. Fungsi `send_message`

Fungsi ini mengimplementasikan proses pengiriman pesan kepada pemain lain. Terdapat 4 nilai masukan pada fungsi ini, yaitu: 2 buah nilai masukan yang sama-sama berupa ID pemain dengan nama *player_one* dan *player_two*, subjek pesan dengan nama *subject* dan isi pesan dengan nama *message*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.18.

```
INSERT INTO player_message
VALUES('',player_one, player_two, msg_subject,
msg);
SELECT 1 AS result, CONCAT("Message Sent") AS
info;
```

Kode Sumber 4.18. Potongan fungsi `send_message`

4.1.2.35. Fungsi `send_trade_request`

Fungsi ini mengimplementasikan proses pengiriman *request* pertukaran kartu kepada pemain lain. Terdapat 2 nilai masukan pada fungsi ini yang sama-sama berupa ID pemain dengan nama

player_one dan *player_two*. Potongan kode implementasi *stored procedure* fungsi ini dapat dilihat pada Kode Sumber 4.19.

```
INSERT INTO trading_detail
VALUES('',player_one, player_two);
SELECT r.RequestID FROM trading_detail r
WHERE r.SenderPlayer=player_one AND
r.RequestedPlayer=player_two
ORDER BY r.RequestID DESC LIMIT 1;
```

Kode Sumber 4.19. Potongan fungsi `send_trade_request`

4.1.2.36. Fungsi `show_battle_rank`

Fungsi ini mengimplementasikan proses pengambilan data keterangan hasil *battle* antar pemain pada permainan.Fungsi ini mempunyai satu nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.37.

4.1.2.37. Fungsi `show_player_ranking`

Fungsi ini mengimplementasikan proses pengambilan data *ranking battle* antar pemain pada permainan. Fungsi ini mempunyai satu nilai masukan berupa ID pemain dengan nama *player_id*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 7.38.

4.2. Implementasi Lapisan Kontrol

Kelas ini merupakan kelas pengendali yang mengatur alur jalannya data pada *web service* ini. Kelas lapisan kontrol memanggil fungsi-fungsi yang ada pada kelas `NuSOAP library` guna membangun *web service*. Pada lapisan kontrol, selain terdapat kelas *library*, juga terdapat kelas `ServiceServer`. Kelas `ServiceServer` bertindak sebagai *web service* penyedia layanan fungsi-fungsi pada *game*. Kelas

ServiceServer memanggil fungsi pada NuSOAP library untuk membangun struktur *web service* dan mengembalikan data kembalian kepada aplikasi pengguna.

```
$this->load->library("Nusoap_Library");
$this->nusoap_server = new soap_server();
$this->nusoap_server-
>configureWSDL('ci_service', 'urn:ci_service');
```

Kode Sumber 4.20. Potongan kode sumber pada ServiceServer

Kelas ServiceServer dapat melakukan instansiasi *soap server* dan melakukan *generate* XML *server* melalui pemanggilan fungsi pada NuSoap library seperti pada Kode Sumber 4.20.

Kelas ServiceServer kemudian melakukan *register* daftar fungsi-fungsi yang ada pada *web service.* Pendaftaran daftar fungsi dilakukan dengan memberikan spesifikasi fungsi tersebut seperti pada Kode Sumber 4.21.

```
$this->nusoap_server->register('player.login',
array('name'=>'xsd:string',
'pass'=>'xsd:string'),array('return'          =>
'xsd:string'),'urn:web_service_agil',
'urn:web_service_agil#login','rpc','encoded','L
ogin');
```

Kode Sumber 4.21. Potongan kode sumber registrasi fungsi

4.3. Implementasi Lapisan Data

Kelas ini mengimplementasikan fungsi-fungsi untuk setiap model yang berhubungan dengan *database*. Berikut uraian implementasi pada lapisan model.

4.3.1. Kelas `Avatar`

Kelas ini merupakan kelas yang berhubungan dengan data *avatar* yang ada pada *database* dan berisi tentang fungsi-fungsi terkait dengan *avatar* pemain.

4.3.2. Kelas `Battle`

Kelas ini merupakan kelas yang berhubungan dengan data *battle* yang ada pada *database*, baik *battle* antar pemain maupun *battle* antara pemain dengan kecerdasan buatan.

4.3.3. Kelas `Building`

Kelas ini merupakan kelas yang berhubungan dengan data *building* pemain pada *database* dan fungsi-fungsi yang terkait dengan data *building* pemain.

4.3.4. Kelas `Card`

Kelas ini merupakan kelas yang berhubungan dengan data kartu yang ada pada *database* dan fungsi-fungsi yang terkait dengan transaksi kartu pada *shop* dan *deck*.

4.3.5. Kelas `Message`

Kelas ini merupakan kelas yang berisi tentang data pesan dan pengiriman pesan antar pemain.

4.3.6. Kelas `Player`

Kelas ini merupakan kelas model yang berisi tentang data-data yang berhubungan dengan pemain. Kelas `player` juga mengimplementasikan fungsi-fungsi yang terkait dengan *player*, *friend*, *login* dan *register*.

4.3.7. Kelas `Request`

Kelas ini merupakan kelas yang berisi tentang data-data permintaan atau *request*, baik itu *request* energi, pertemanan, ataupun pertukaran kartu antar pemain. Kelas *request* juga mengimplementasikan fungsi-fungsi terkait data *request* yang ada pada *database*.

4.3.8. Kelas `MonsterQuest`

Kelas ini merupakan kelas yang berisi tentang data-data *quest* pemain beserta *monster-monster* yang ada pada *game*. Kelas ini juga mengimplementasikan fungsi-fungsi terkait *quest* pemain selama permainan.

4.4. Implementasi Lapisan Antarmuka

Lapisan antarmuka merupakan lapisan yang berhubungan langsung dengan pengguna dan hanya dapat diakses oleh kelas-kelas pada lapisan kontrol. Kelas pada lapisan antarmuka bertugas untuk menampilkan data hasil eksekusi fungsi pada *web service* kepada pengguna.

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada *web service* yang telah dibangun. Pengujian yang dilakukan adalah pengujian terhadap kebutuhan fungsionalitas sistem. Pengujian fungsionalitas mengacu pada fungsi-fungsi yang telah disebutkan pada bab tiga. Pengujian kegunaan program dilakukan dengan mengetahui tanggapan dari pengguna terhadap sistem. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada bagian akhir bab ini.

5.1. Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan tugas akhir ini dilakukan pada lingkungan dan alat kakas sebagai berikut:

Komputer 1

Prosesor	: Intel Core i5 4200M CPU @ 2.50GHz
Memori	: 4.00 GB
Jenis Device	: Laptop
Sistem Operasi	: Microsoft Windows 8.1 64 bit

Komputer 2

Prosesor	: Intel Core i3 M370 CPU @ 2.40GHz
Memori	: 4.00 GB
Jenis Device	: Laptop
Sistem Operasi	: Microsoft Windows 7 Ultimate 32 bit

5.2. Skenario Pengujian

Pada bagian ini akan dijelaskan tentang skenario pengujian yang dilakukan. Pengujian dilakukan dalam 2 bagian. Bagian pertama

yaitu pengujian *stored procedure* pada *layer database* sedangkan pengujian kedua adalah pengujian fungsi-fungsi pada *web service.* Aspek yang diujikan adalah *correctness*, *integrity*, dan *performance*.

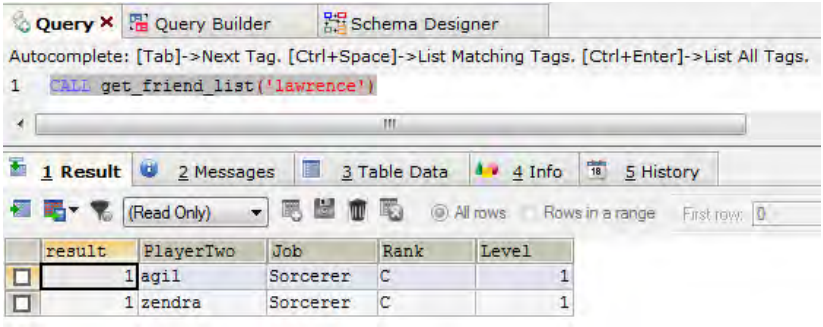
5.2.1. Pengujian *Stored Procedure* pada *Layer Database*

Pengujian ini dilakukan terhadap *stored procedure* yang ada pada *layer database*. Pengujian dilakukan secara *black box* dengan memasukkan data skenario uji coba terhadap *stored procedure* yang ada. Pengujian ini dilakukan terhadap beberapa fungsi *stored procedure* dengan beberapa skenario masukan data yang berbeda. Uraian skenario uji coba dapat dilihat pada Tabel 5.1.

Tabel 5.1. Pengujian *Stored Procedure get_friend_list* Skenario 1

ID	UJ.UC-0001
Nama	Pengujian *stored procedure* menampilkan daftar teman yang dimiliki oleh seorang pemain.
Tujuan Pengujian	Menguji fungsi untuk menampilka daftar teman dari seorang pemain.
Skenario	Eksekusi *stored procedure* menghasilkan satu atau lebih data hasil *select.*
Langkah Pengujian	*Store procedure* dipanggil dengan menggunakan *parameter* ID pemain yang telah memiliki teman.
Hasil Yang Diharapkan	*Stored procedure* mengembalikan daftar teman yang dimiliki oleh pemain.
Hasil Yang Didapat	Daftar teman yang dimiliki oleh pemain.
Hasil Pengujian	Berhasil

Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.1



Gambar 5.1. Hasil uji coba skenario 1

Pada uji coba skenario 1, fungsi *store procedure* menghasilkan satu atau lebih data hasil *select* dengan nilai masukan nama pemain yang *valid.* Untuk uji coba dengan skenario lain dapat dilihat pada Lampiran B. Hasil pengujian.

5.2.2. Pengujian Fungsi pada *Web Service*

Pengujian fungsi pada *web service* pada aspek *correctness* dilakukan dengan menyiapkan sejumlah skenario sebagai tolak ukur keberhasilan pengujian. Pengujian fungsi pada *web service* mengambil data *friend list* dan data *player card* dari seorang pemain. Uraian uji coba skenario dapat dilihat pada Tabel 5.2.

Tabel 5.2. Pengujian Fungsi *get_friend_list*
Skenario 1

ID	UJ.UC-0011
Nama	Pengujian fungsi pada *web service* menampilkan daftar teman yang dimiliki oleh seorang pemain.

Tujuan Pengujian	Menguji fungsi untuk menampilkan daftar teman dari seorang pemain.
Skenario	Eksekusi fungsi pada *web service* menghasilkan satu atau lebih data hasil *select.*
Langkah Pengujian	Fungsi pada *web service* dipanggil dengan menggunakan *parameter* ID pemain yang telah memiliki teman.
Hasil Yang Diharapkan	Fungsi pada *web service* mengembalikan daftar teman yang dimiliki oleh pemain.
Hasil Yang Didapat	Daftar teman yang dimiliki oleh pemain.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.2.

Gambar 5.2. Hasil Uji Coba Fungsi Skenario 1

Gambar 5.2 menunjukkan hasil eksekusi fungsi *get_friend_list* pada *web service*. Fungsi mengembalikan *query result* kepada pengguna dalam bentuk data XML yang berisi kumpulan data *friend* yang dimiliki oleh pemain.

5.2.3. Pengujian Integrasi Antar Modul

Pengujian integrasi antar modul dengan *web service* dilakukan pada aplikasi pengguna. Berikut beberapa skenario uji coba integrasi *web service* dengan permainan.

5.2.3.1. Uji Coba Proses *Login*

Proses *login* diambil sebagai skenario uji coba karena proses *login* merupakan salah satu fungsi umum yang sering digunakan oleh pengguna. Pengujian proses *login* dilakukan pada aplikasi pengguna dengan mengisi halaman *form login* yang disediakan pada permainan seperti yang ditunjukkan pada Gambar 5.3.

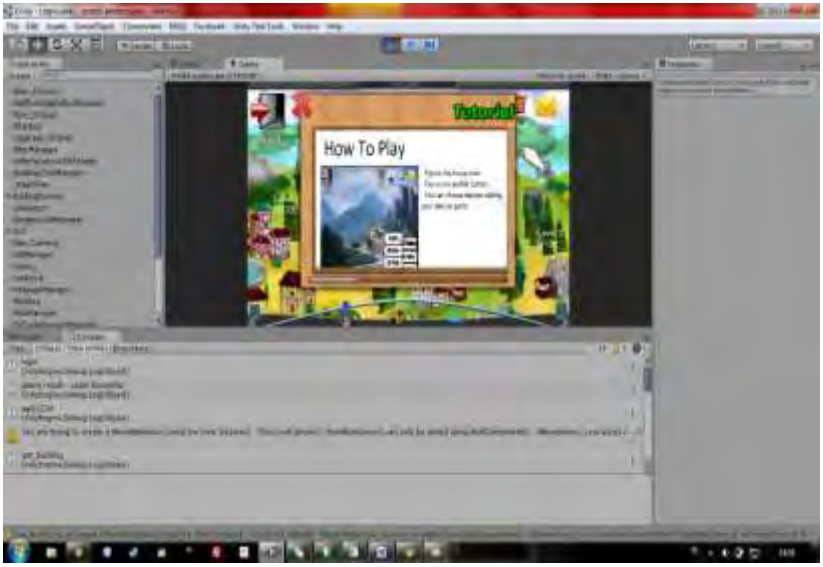
Gambar 5.3. Halaman *login* pada *game*

Gambar 5.3 menunjukkan halaman *login* ke dalam permainan. Uji coba skenario *login* pada permainan ditunjukkan pada Tabel 5.3.

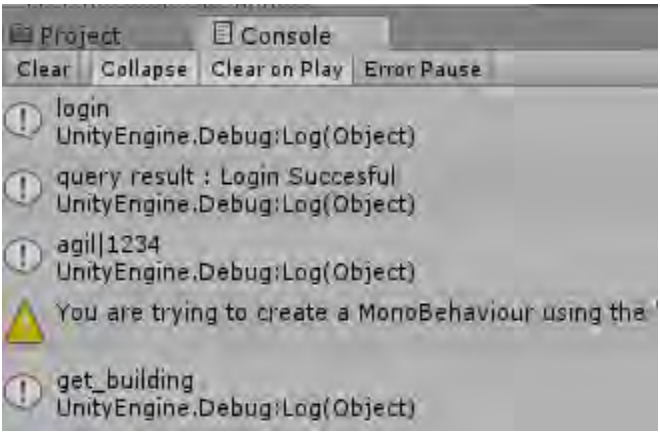
Tabel 5.3. Uji Coba Skenario *Login* pada Permainan

ID	UJ.UC-0021
Nama	Pengujian fungsi pada *web service* pada aplikasi permainan dengan fungsi *login*.
Tujuan Pengujian	Menguji fungsi untuk proses *login* pemain ke dalam permainan.
Skenario	Eksekusi fungsi *login web service* pada aplikasi permainan menghasilkan hasil eksekusi proses *login*.
Langkah Pengujian	Fungsi pada *web service* dipanggil dengan menggunakan *parameter* ID pemain dan *password* yang dimasukkan oleh pemain.
Hasil Yang Diharapkan	Fungsi pada *web service* mengembalikan hasil eksekusi proses *login* ke aplikasi pengguna.

Hasil Yang Didapat	Hasil eksekusi proses *login* dari *web service*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.5.



Gambar 5.4. Tampilan *Home* pada Saat *In-Game*



Gambar 5.5. Tampilan Hasil Eksekusi Fungsi *web service* pada permainan

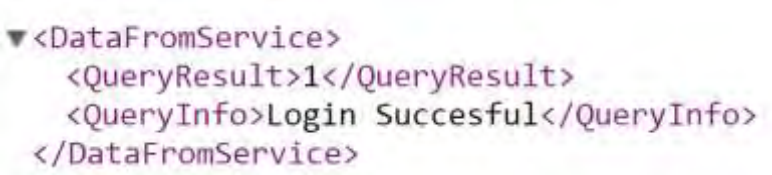
Pada gambar Gambar 5.4 ditunjukkan halaman *home* pada permainan setelah pemain berhasil *login* ke dalam permainan. Gambar 5.5 menunjukkan tampilan *message* yang dikirim oleh *web service* kepada aplikasi pengguna yang menunjukkan bahwa pemain telah berhasil melakukan proses *login.*

Uji coba juga dilakukan pada *web service* secara *independent* dengan cara pengaksesan fungsi *web service* melalui alamat URL pada *browser*. Uji coba skenario *login* pada *browser* ditunjukkan pada Tabel 5.4.

Tabel 5.4. Uji Coba Skenario *Login* pada *browser*

ID	UJ.UC-0031
Nama	Pengujian fungsi pada *web service* pada *browser* dengan fungsi *login*.
Tujuan Pengujian	Menguji fungsi untuk proses *login* pemain.

Skenario	Eksekusi fungsi *login web service* pada browser menghasilkan hasil eksekusi proses *login*.
Langkah Pengujian	Fungsi pada *web service* dipanggil dengan menggunakan *parameter* ID pemain dan *password* yang dimasukkan oleh pemain.
Hasil Yang Diharapkan	Fungsi pada *web service* mengembalikan hasil eksekusi proses *login* ke pengguna.
Hasil Yang Didapat	Hasil eksekusi proses *login* dari *web service*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.6.



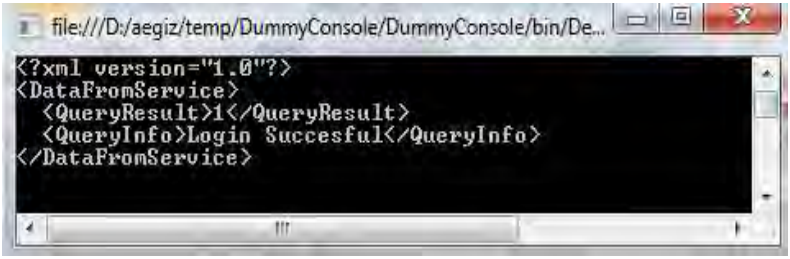
Gambar 5.6. Hasil uji coba fungsi *login* pada *browser*

Uji coba lain dilakukan dengan pengaksesan fungsi yang sama pada *web service* melalui *C# console*. Uji coba skenario ini dapat dilihat pada Tabel 5.5.

Tabel 5.5. Uji Coba Skenario *Login* pada *console*

ID	UJ.UC-0041
Nama	Pengujian fungsi pada *web service* pada *console* dengan fungsi *login*.

Tujuan Pengujian	Menguji fungsi untuk proses *login* pemain.
Skenario	Eksekusi fungsi *login web service* pada console menghasilkan hasil eksekusi proses *login.*
Langkah Pengujian	Fungsi pada *web service* dipanggil dengan menggunakan *parameter* ID pemain dan *password* yang dimasukkan oleh pemain.
Hasil Yang Diharapkan	Fungsi pada *web service* mengembalikan hasil eksekusi proses *login* ke pengguna.
Hasil Yang Didapat	Hasil eksekusi proses *login* dari *web service*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.7.



Gambar 5.7. Hasil Uji Coba Pengaksesan Fungsi *login* melalui *console*

5.2.3.2. Uji Coba Fungsi *Edit Deck*

Fungsi *edit deck* berhubungan dengan data kartu pemain, di mana kartu adalah mekanisme utama pada permainan ini, maka dari

itulah fungsi ini diambil sebagai skenario uji coba. Pengguna memasuki menu *edit deck* dan pemain dapat mengatur kartu apa saja yang akan dibawa pada pertarungan. Skenario uji coba dituliskan pada Tabel 5.6.

Tabel 5.6. Uji Coba Skenario *Edit Deck*

ID	UJ.UC-0051
Nama	Pengujian fungsi *edit deck* pada permainan.
Tujuan Pengujian	Menguji fungsi untuk proses pengaturan kartu pemain pada *deck*.
Skenario	Eksekusi menu *edit deck* pada permainan.
Langkah Pengujian	Pemain memasukkan kartu baru ke dalam *deck* dan memindahkan kartu dari *deck* ke *trunk*.
Hasil Yang Diharapkan	*Deck* pemain dapat tersimpan dan pemain dapat menggunakan kartu pada pertarungan.
Hasil Yang Didapat	*Deck* pemain yang telah tersimpan dengan kartu kartu yang baru dimasukkan sebelumnya.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.8 dan Gambar 5.9.

Gambar 5.8. Hasil Uji Coba Proses *Edit Deck*

Gambar 5.9. Hasil Uji Coba Proses *Edit Deck* pada Pertarungan

Hasil uji coba proses *edit deck* dapat dilihat pada Gambar 5.8 di mana pemain mengatur kartu apa saja yang dimasukkan ke dalam

deck dan pemain dapat mengakses kartu-kartu tersebut saat pertarungan yang dapat dilihat pada Gambar 5.9.

5.2.3.3. Uji Coba Proses *Buy Card*

Pemain dapat membeli kartu di toko atau *shop*, dan memungkinkan pemain untuk sering membeli kartu pada *shop*, karena itu proses ini diambil sebagai contoh uji coba. Untuk skenario pada uji coba ini dapat dilihat pada Tabel 5.7.

Tabel 5.7. Skenario Uji Coba Proses *Buy Card*

ID	UJ.UC-0061
Nama	Pengujian fungsi *buy card*.
Tujuan Pengujian	Menguji fungsi untuk proses pembelian kartu.
Skenario	Eksekusi menu *buy card* pada *shop*.
Langkah Pengujian	Pemain memilih kartu pada *shop* dan melakukan proses pembelian dengan uang yang tidak mencukupi.
Hasil Yang Diharapkan	*Shop* menolak pembelian pemain karena uang yang tidak mencukupi.
Hasil Yang Didapat	Pemain tidak berhasil membeli kartu karena uang yang tidak mencukupi.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.10.

Gambar 5.10. Uji Coba Proses *Buy Card*

Pada Gambar 5.10 d**i**tunjukkan bahwa pemain hendak membeli kartu namun dengan uang yang tidak mencukupi. Karena uang tidak mencukupi, *shop* menolak pembelian kartu oleh pemain dengan mengembalikan keterangan *not enough money*.

5.2.3.4. Uji Coba Proses *Trading Card*

Proses *trading card* diambil sebagai salah satu contoh uji coba karena pada proses ini melibatkan 2 orang pemain dan data kartu antar pemain dengan proses yang cukup banyak. Skenario pada uji coba ini dapat dilihat pada Tabel 5.8.

Tabel 5.8. Skenario Uji Coba Proses *Trading Card*

ID	UJ.UC-0071
Nama	Pengujian fungsi *trading card*
Tujuan Pengujian	Menguji fungsi untuk proses pertukaran kartu antar pemain.
Skenario	Eksekusi menu *trading card* antar pemain.
Langkah Pengujian	Pemain memilih kartu yang hendak ditukarkan dengan kartu pemain lain, dan mengirim permintaan pertukaran kartu kepada pemain tersebut.
Hasil Yang Diharapkan	Kartu antar pemain berhasil ditukar.
Hasil Yang Didapat	Kartu antar pemain berhasil ditukar.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 5.11 dan Gambar 5.12.

Gambar 5.11. Uji Coba Proses *Trading Card*

Gambar 5.12. Uji Coba Proses Penerimaan *Request Trading Card*

Pada Gambar 5.11 dan Gambar 5.12 ditunjukkan bahwa seorang pemain dapat mengirim permintaan pertukaran kartu kepada pemain lain. Pemain tersebut kemudian dapat menerima permintaan tersebut untuk menukar kartu yang telah disepakati bersama.

5.3. Uji Coba *Performance*

Untuk uji coba *performance* digunakan perangkat uji coba Apache Jmeter 2.11. Untuk *web server* yang digunakan menggunakan kriteria sebagai berikut.

- PHP versi 5.4.30
- MySQL versi 5.5.40-cll
- Apache versi 2.2.27

Uji coba dilakukan dengan melakukan pengaksesan *web service* melalui *virtual client* dari aplikasi Apache Jmeter 2.11. Dimulai dengan 100 klien, sampai dengan 700 klien, dengan peningkatan per 100 klien. Uji coba dilakukan pada 2 buah komputer berbeda. Hasil uji coba pada komputer pertama dapat dilihat pada Tabel 5.9 dan hasil uji coba pada komputer kedua dapat dilihat pada Tabel 5.10 dengan keterangan atribut tabel sebagai berikut.

- *Samples.* Jumlah klien uji coba saat ini.
- *Avg*. Waktu rata-rata yang diperlukan untuk menyelesaikan uji coba.
- *Min.* Waktu minimal yang diperlukan untuk menyelesaikan uji coba.
- *Max.* Waktu maksimal yang diperlukan untuk menyelesaikan uji coba.
- *Error*. Presentase tingkat kesalahan yakni presentase kegagalan *request* oleh klien.
- *Throughput.* Jumlah *request* yang dilakukan setiap detik.
- *KB/sec.* Tingkat kecepatan *request* klien per detik.

Tabel 5.9. Hasil Uji Coba Performa Komputer Pertama

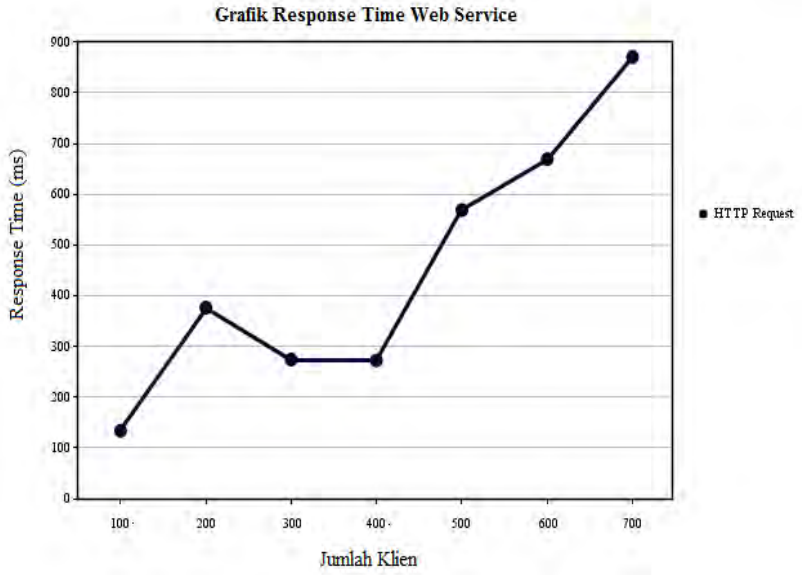
Samples	Avg	Min	Max	Error	Throughput	KB/sec
100	56	22	88	0,00%	49,4/sec	13,36
200	302	29	3062	0,00%	40,2/sec	10,88
300	885	29	3092	0,00%	59,4/sec	16,08
400	1361	53	9031	0,00%	39,9/sec	10,79
500	2557	47	9290	0,00%	44,7/sec	12,10
600	2733	42	20995	0,17%	27,0/sec	7,38
700	2068	84	21008	0,29%	22,5/sec	13,41

Tabel 5.10. Hasil Uji Coba Performa Komputer Kedua

Samples	Avg	Min	Max	Error	Throughput	KB/sec
100	98	75	181	0,00%	47,6/sec	18,27
200	624	98	4610	0,00%	40,3/sec	15,32
300	1078	102	7145	0,00%	33,9/sec	12,95
400	2996	98	10539	0,00%	32,8/sec	12,21
500	3465	117	22525	0,00%	21,4/sec	12,31
600	3427	99	21003	0,67%	27,3/sec	10,63
700	3336	40	21005	0,86%	29,9/sec	8,60

Pada Tabel 5.9 dan Tabel 5.10 ditunjukkan hasil uji coba performa pada kedua komputer, dimulai dari uji coba dengan 100 klien sampai dengan 700 klien dengan kenaikan tiap tahap 100 klien. Pada kedua buah komputer terlihat untuk performa *web service* masih terlihat bagus sampai dengan uji coba dengan 500 klien, dengan ditandainya tingkat *error* sejumlah 0,00%. Namun untuk jumlah klien sebanyak 600 dan 700, pada masing-masing komputer mengalami kenaikan *error*, yakni pada komputer pertama sebanyak 0,17% pada tingkat 600 klien dan sebanyak 0,29% pada tingkat 700 klien. Sedangkan pada komputer kedua, persentase *error* meningkat menjadi 0,67% pada tingkat 600 klien, dan sejumlah 0,86% pada tingkat 700 klien.

Uji coba performa juga dilakukan untuk melihat performa *response time* pada *web service*. Sama seperti skenario pengujian untuk melihat tingkat *error* sebelumnya, uji coba dimulai dengan 100 klien, dengan pertambahan 100 klien sampai dengan 700 klien. Hasil uji coba ini dapat dilihat pada Gambar 5.13.



Gambar 5.13. Grafik Hasil Uji Coba *Response Time*

Pada Gambar 5.13 ditunjukkan mengenai hasil uji coba *response time web service* untuk setiap jumlah klien mulai dari 100 klien sampai 700 klien dengan kenaikan 100 klien. Dari grafik tersebut dapat terlihat bahwa untuk *request* yang dilakukan klien kepada *server* memerlukan waktu yang relatif singkat untuk klien sejumlah 100 sampai dengan 400, namun *response time* sedikit demi sedikit meningkat seiring bertambahnya jumlah klien.

5.4. *Evaluasi Pengujian*

Pada subbab ini akan diberikan hasil evaluasi dari pengujian-pengujian yang telah dilakukan. Evaluasi yang diberikan meliputi evaluasi pengujian kebutuhan fungsional.

5.4.1. Evaluasi Pengujian Fungsionalitas

Rangkuman mengenai hasil pengujian *stored procedure* dan fungsi pada *web service* dapat dilihat pada Tabel 5.11. Berdasarkan data pada tabel tersebut, semua skenario pengujian berhasil dan program berjalan dengan baik. Sehingga bisa ditarik disimpulkan bahwa fungsionalitas dari program telah bekerja sesuai dengan yang diharapkan.

Tabel 5.11. Rangkuman Hasil Pengujian

ID	Nama	Skenario	Hasil
UJ.UC-0001	Pengujian *stored procedure* menampilkan daftar teman yang dimiliki oleh seorang pemain.	Eksekusi fungsi pada *web service* menghasilkan satu atau lebih data hasil *select.*	Berhasil
UJ.UC-0002	Pengujian *stored procedure* menampilkan daftar teman yang dimiliki oleh seorang pemain.	Eksekusi *stored procedure* menghasilkan data yang kosong.	Berhasil
UJ.UC-0003	Pengujian *stored procedure* menampilkan daftar kartu yang dimiliki oleh seorang pemain.	Eksekusi fungsi pada *web service* menghasilkan satu atau lebih data hasil *select.*	Berhasil
UJ.UC-0004	Pengujian *stored procedure* menampilkan daftar kartu yang dimiliki oleh seorang pemain.	Eksekusi *stored procedure* menghasilkan data yang kosong.	Berhasil
UJ.UC-0011	Pengujian fungsi pada *web service* menampilkan daftar teman yang dimiliki oleh seorang pemain.	Eksekusi fungsi pada *web service* menghasilkan satu atau lebih data hasil *select.*	Berhasil
UJ.UC-0012	Pengujian fungsi pada *web service* menampilkan daftar kartu yang dimiliki oleh seorang pemain.	Eksekusi fungsi pada *web service* menghasilkan data kosong.	Berhasil

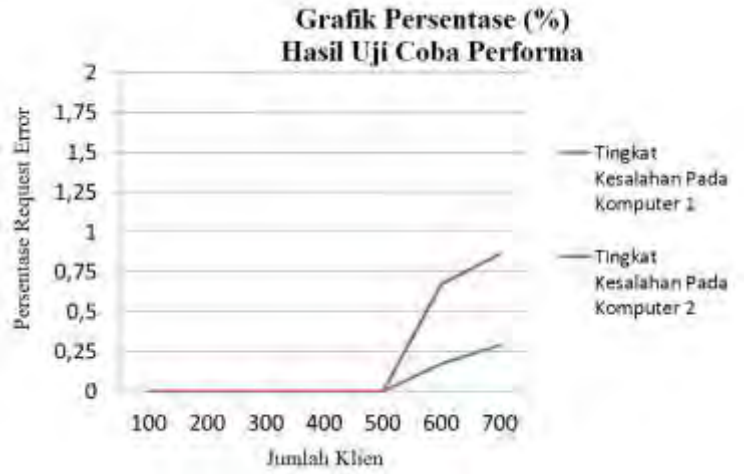
ID	Nama	Skenario	Hasil
UJ.UC-0021	Pengujian fungsi pada *web service* pada aplikasi permainan dengan fungsi *login*.	Eksekusi fungsi *login web service* pada aplikasi permainan menghasilkan hasil eksekusi proses *login*.	Berhasil
UJ.UC-0031	Pengujian fungsi pada *web service* pada *browser* dengan fungsi *login*.	Eksekusi fungsi *login web service* pada *browser* menghasilkan hasil eksekusi proses *login*.	Berhasil
UJ.UC-0041	Pengujian fungsi pada *web service* pada *console* dengan fungsi *login*.	Eksekusi fungsi *login web service* pada *console* menghasilkan hasil eksekusi proses *login*.	Berhasil
UJ.UC-0051	Pengujian fungsi *edit deck* pada permainan.	Eksekusi menu *edit deck* pada permainan.	Berhasil
UJ.UC-0061	Pengujian fungsi *buy card* pada permainan.	Eksekusi menu *buy card* pada *shop* pada permainan.	Berhasil
UJ.UC-0071	Pengujian fungsi *trading card* antar pemain pada permainan.	Eksekusi menu *trading card* pada permainan.	Berhasil

Pada Tabel 5.11 diuraikan mengenai skenario uji coba fungsionalitas yang telah dilakukan. Uji coba mencakup uji coba *stored procedure*, dan uji coba integrasi *web service* dengan modul-modul lain yang ada pada permainan.

5.4.2. Evaluasi Pengujian Performa

Pada hasil uji coba performa yang telah dilakukan dapat terlihat bahwa *web service* dapat melayani sekitar 500 klien sekaligus,

namun performa berkurang ketika mencapai angka 600 klien. Untuk hasil uji coba performa dapat dilihat pada Gambar 5.14.



Gambar 5.14. Grafik Hasil Uji Coba Performa *Web Service*

Hasil uji coba *performance* digambarkan dengan grafik pada Gambar 5.14. Terlihat pada grafik di atas, untuk uji coba mulai dari 100 klien sampai dengan 500 klien tidak terdapat kesalahan (*error*) di mana kesalahan terjadi apabila ada klien yang tidak terlayani oleh *web service*. Namun kesalahan terjadi pada uji coba selanjutnya dengan klien yang lebih banyak yakni 600 klien di mana terdapat tingkat kesalahan sebanyak 0,17% (1 klien) yang tidak terlayani pada komputer 1, sedangkan pada komputer 2 tingkat kesalahan naik menjadi 0,67% (4 klien). Kemudian pada uji coba dengan 700 klien muncul tingkat kesalahan sebanyak 0,29% (2 klien) pada komputer 1, dan sebanyak 0,86% (6 klien) yang tidak terlayani pada komputer 2. Tingkat *error* dimungkinkan muncul karena tingkat kecepatan pengaksesan klien yang menurun saat uji coba 600 dan 700 klien, di mana jumlah *request* yang dapat dikirim oleh komputer juga bisa

berkurang, mengakibatkan munculnya *virtual client* yang gagal melakukan *request* terhadap *web service*.

Sementara untuk uji coba performa dengan cara *response time* dapat dikatakan bahwa performa *web service* cukup baik dengan ditandainya *response time* yang tidak melebihi 1000 ms untuk jumlah klien sebanyak 700.

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan tugas akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1. Kesimpulan

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut:

1. Aturan main pada permainan *Card Warlock Saga* dapat diimplementasikan menjadi fungsi-fungsi yang dibungkus dalam *web service* dan dapat diakses oleh pengguna melalui aplikasi permainan.
2. *Web service* yang telah dibangun dapat terintegrasi dengan modul-modul lain yang ada dalam permainan *Card Warlock Saga* dengan baik.
3. Sistem dapat diuji secara *independent* (pada *layer database*, *layer* dan *browser*) maupun bersama-sama dengan aplikasi permainan.

6.2. Saran

Berikut saran-saran untuk pengembangan dan perbaikan sistem di masa yang akan datang. Di antaranya adalah sebagai berikut:

1. Meningkatkan efisiensi program dengan membagi-bagi fungsi berdasarkan kategori dan jenisnya masing-masing.
2. Menambahkan fitur untuk menampilkan hasil eksekusi fungsi pada *web service* ke dalam permainan.

LAMPIRAN A. KODE SUMBER

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `accept_energy_request`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`accept_energy_request`(player_one VARCHAR(50), player_two
VARCHAR(50))
BEGIN
    DECLARE energy_left INT;
    DECLARE request_id INT;
    DECLARE is_valid VARCHAR(1);
    SELECT 1 INTO is_valid FROM player_request
        WHERE RequestedPlayer=player_one AND
SenderPlayer=player_two
        AND Information='Energy Sent'
        LIMIT 1;
    SELECT Energy INTO energy_left FROM players WHERE
PlayerID=player_one;
    IF(is_valid!="") THEN
        IF(energy_left>=25) THEN
            SELECT 0 AS result, CONCAT("Your current energy
is full, you can accept this extra energy later") AS info;
        ELSE
            UPDATE players SET Energy=energy_left+1 WHERE
PlayerID=player_one;
            SELECT requestid INTO request_id FROM
player_request
                WHERE RequestedPlayer=player_one AND
SenderPlayer=player_two
                AND Information='Energy Sent'
                ORDER BY RequestTime ASC LIMIT 1;
            DELETE FROM player_request WHERE
RequestID=request_id;
            SELECT 1 AS result, CONCAT("Extra energy
received") AS info;
        END IF;
    ELSE
        SELECT -1 AS result, CONCAT("No Request Found") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.1. *Stored Procedure* `accept_energy_request`

```
DELIMITER $$
USE `yowanda1_cws`$$
```

```
DROP PROCEDURE IF EXISTS `accept_friend_request`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`accept_friend_request`(player_one VARCHAR(50), player_two
VARCHAR(50))
BEGIN
    DECLARE is_valid VARCHAR(1);
    SELECT 1 INTO is_valid FROM player_request
    WHERE SenderPlayer=player_two AND
RequestedPlayer=player_one AND Information='friend request
sent';
    IF(is_valid!="") THEN
        INSERT INTO player_friend VALUES('',player_one,
player_two);
        INSERT INTO player_friend VALUES('',player_two,
player_one);
        DELETE FROM player_request
        WHERE SenderPlayer=player_two AND
RequestedPlayer=player_one AND Information='friend request
sent';
        SELECT 1 AS result, CONCAT("You and ", player_two, "
are now friends") AS info;
    ELSE
        SELECT 0 AS result, CONCAT("No Friend Request
Found") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.2. ***Stored Procedure*** **`accept_friend_request`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `clear_deck`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`clear_deck`(player_id VARCHAR(50))
BEGIN
    UPDATE player_card SET DeckQuantity=0,
TrunkQuantity=TotalQuantity WHERE PlayerID=player_id;
    SELECT 1 AS result, CONCAT("Deck is Cleared") AS info;
    END$$
DELIMITER ;
```

Kode Sumber 7.3. ***Stored Procedure*** **`clear_deck`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `decline_trade_request`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
```

```
`decline_trade_request`(req_id INT)
BEGIN
    DECLARE is_valid INT;
    SELECT 1 INTO is_valid FROM trading_detail WHERE
RequestID=req_id LIMIT 1;
    IF(is_valid!="") THEN
        DELETE FROM trading_detail WHERE RequestID=req_id;
        SELECT 1 AS result, CONCAT("Trade Request Declined")
AS info;
    ELSE
        SELECT 0 AS result, CONCAT("Invalid ID") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.4. *Stored Procedure* `decline_trade_request`

```
DELIMITER $$
USE `yowanda1_cws`$
DROP PROCEDURE IF EXISTS `edit_avatar`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`edit_avatar`(player_id VARCHAR(50), head_slot VARCHAR(50),
eye_slot VARCHAR(50), mouth_slot VARCHAR(50))
BEGIN
    DECLARE item_id VARCHAR(10);
    DELETE FROM player_avatar WHERE PlayerID=player_id;
    SELECT av.ItemID INTO item_id FROM avatar_item av WHERE
av.Name=head_slot;
    INSERT INTO player_avatar VALUES('',player_id, item_id);
    SELECT av.ItemID INTO item_id FROM avatar_item av WHERE
av.Name=eye_slot;
    INSERT INTO player_avatar VALUES('',player_id, item_id);
    SELECT av.ItemID INTO item_id FROM avatar_item av WHERE
av.Name=mouth_slot;
    INSERT INTO player_avatar VALUES('',player_id, item_id);
    SELECT 1 AS result, CONCAT("Avatar has been updated") AS
info;
    END$$
DELIMITER ;
```

Kode Sumber 7.5. *Stored Procedure* `edit_avatar`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_battle_list`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`get_battle_list`(dungeon_id VARCHAR(10))
```

```
BEGIN
    DECLARE is_empty INT;
    SELECT 1 INTO is_empty FROM quest_battle WHERE
DungeonID=dungeon_id LIMIT 1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, BattleID FROM quest_battle WHERE
DungeonID=dungeon_id;
    ELSE
        SELECT 0 AS result, CONCAT("Invalid DungeonID") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.6. ***Stored Procedure*** **`get_battle_list`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_card_request_list`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`get_card_request_list`(req_id INT)
BEGIN
    DECLARE is_valid INT;
    SELECT 1 INTO is_valid FROM trade_request_detail WHERE
RequestID=req_id LIMIT 1;
    IF(is_valid!="") THEN
        SELECT 1 AS result, t.PlayerID, c.Name, t.Quantity
        FROM trade_request_detail t INNER JOIN cards c ON
t.CardID=c.CardID
        WHERE t.RequestID=req_id ORDER BY t.TableID ASC;
    ELSE
        SELECT 0 AS result, CONCAT("Invalid ID") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.7. ***Stored Procedure*** **`get_card_request_list`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_energy_request_list`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`get_energy_request_list`(player_id VARCHAR(50))
BEGIN
    DECLARE is_valid VARCHAR(1);
    SELECT 1 INTO is_valid FROM player_request WHERE
RequestedPlayer=player_id AND Information='Energy Sent'
```

```
LIMIT 1;
    IF(is_valid!="") THEN
        SELECT 1 AS result, SenderPlayer AS player FROM
player_request     WHERE     RequestedPlayer=player_id     AND
Information='Energy Sent';
    ELSE
        SELECT 0 AS result, CONCAT("Data Not Found") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.8. *Stored Procedure* `get_energy_request_list`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_friend_list`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_friend_list`(player_id VARCHAR(50))
BEGIN
    DECLARE is_empty INT;
    SELECT  1  INTO  is_empty  FROM  player_friend  WHERE
PlayerOne=player_id LIMIT 1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, p.PlayerTwo, q.Job,
        q.Rank, q.Level FROM player_friend p
        INNER JOIN players q ON p.PlayerTwo=q.PlayerID
        WHERE p.PlayerOne=player_id;
    ELSE
        SELECT 0 AS result, CONCAT("You Have No Friends Yet,
Add Your Friends!") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.9. *Stored Procedure* `get_friend_list`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_friend_request_list`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_friend_request_list`(player_id VARCHAR(50))
BEGIN
    DECLARE is_valid VARCHAR(1);
    SELECT  1  INTO  is_valid  FROM  player_request  WHERE
RequestedPlayer=player_id  AND  Information='Friend  Request
Sent' LIMIT 1;
    IF(is_valid!="") THEN
```

```
        SELECT 1 AS result, p.PlayerID, p.Job, p.Rank,
p.Level
        FROM player_request r INNER JOIN players p ON
p.PlayerID=r.SenderPlayer
        WHERE RequestedPlayer=player_id AND
Information='friend request sent';
    ELSE
        SELECT 0 AS result, CONCAT("Data Not Found") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.10. ***Stored Procedure*** **`get_friend_request_list`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_list_avatar`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`get_list_avatar`(player_id varchar(50))
BEGIN
    DECLARE gender VARCHAR(10);
    SELECT p.Gender INTO gender FROM players p WHERE
PlayerID=player_id;
    SELECT av.Name, av.Slot FROM avatar_item av WHERE
av.Gender=gender;
    END$$
DELIMITER ;
```

Kode Sumber 7.11. ***Stored Procedure*** **`get_list_avatar`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_messages`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`get_messages`(player_id VARCHAR(50))
BEGIN
    DECLARE is_empty VARCHAR(1);
    SELECT 1 INTO is_empty FROM player_message WHERE
PlayerReceiver=player_id LIMIT 1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, m.PlayerSender AS Sender,
m.Subject, m.Message FROM player_message m WHERE
m.PlayerReceiver=player_id;
    ELSE
        SELECT 0 AS result, CONCAT("No incoming messages")
AS info;
    END IF;
```

```
    END$$
DELIMITER ;
```

Kode Sumber 7.12. ***Stored Procedure*** **`get_messages`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_monster_list`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_monster_list`(battle_id VARCHAR(10))
BEGIN
    DECLARE is_empty INT;
    SELECT 1 INTO is_empty FROM battle_monster b INNER JOIN
monsters     m     ON     b.MonsterID=m.MonsterID     WHERE
b.BattleID=battle_id LIMIT 1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, m.Name FROM battle_monster b
        INNER JOIN monsters m ON b.MonsterID=m.MonsterID
        WHERE b.BattleID=battle_id;
    ELSE
        SELECT 0 AS result, CONCAT("Invalid Battle ID") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.13. ***Stored Procedure*** **`get_monster_list`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_name_by_fb`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_name_by_fb`(player_fb_id VARCHAR(50))
BEGIN
    DECLARE is_valid INT;
    SELECT    1    INTO    is_valid    FROM    players    WHERE
FB_ID=player_fb_id LIMIT 1;
    IF(is_valid!="") THEN
        SELECT 1 AS result, p.PlayerID
        FROM players p WHERE p.FB_ID=player_fb_id;
    ELSE
        SELECT 0 AS result, CONCAT("FB Doesn't Exist") AS
info;
    END IF;
    END$$ DELIMITER ;
```

Kode Sumber 7.14. ***Stored Procedure*** **`get_name_by_fb`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_partial_profile`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_partial_profile`(player_id VARCHAR(50))
BEGIN
    DECLARE is_valid INT;
    SELECT    1    INTO    is_valid    FROM    players    WHERE
PlayerID=player_id;
    IF(is_valid!="") THEN
        SELECT  1  AS  result,  p.PlayerID,  p.Job,  p.Rank,
p.Level
        FROM players p WHERE p.PlayerID=player_id;
    ELSE
        SELECT  0  AS  result,  CONCAT("Data  Could  Not  Be
Found") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.15. *Stored Procedure* `get_partial_profile`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_player_avatar`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_player_avatar`(player_id VARCHAR(50))
BEGIN
    SELECT b.Slot, b.Name FROM player_avatar a INNER JOIN
avatar_item      b      ON      b.ItemID=a.ItemID      WHERE
a.PlayerID=player_id;
    END$$
DELIMITER ;
```

Kode Sumber 7.16. *Stored Procedure* `get_player_avatar`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_player_building`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_player_building`(player_id VARCHAR(50))
BEGIN

    SELECT b.Name, b.Level, b.MaxXP, b.CurrentXP,
    b.ProductionQuantity, b.ProductClaimed, b.PrefabName
    FROM player_building b WHERE b.PlayerID=player_id ORDER
BY b.Name;
```

```
    END$$
DELIMITER ;
```

Kode Sumber 7.17. *Stored Procedure* `get_player_building`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_player_cards`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_player_cards`(player_id varchar(50))
BEGIN
    DECLARE is_empty INT;
    SELECT 1 INTO is_empty FROM player_card a INNER JOIN
cards b ON a.CardID=b.CardID WHERE PlayerID=player_id LIMIT
1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, b.Name, a.TotalQuantity AS
quantity FROM player_card a INNER JOIN cards b ON
a.CardID=b.CardID WHERE PlayerID=player_id;
    ELSE
        SELECT 0 AS result, CONCAT("Trunk is Empty") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.18. *Stored Procedure* `get_player_cards`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_player_deck`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_player_deck`(player_id VARCHAR(50))
BEGIN
    DECLARE is_empty INT;
    SELECT 1 INTO is_empty FROM player_card WHERE
PlayerID=player_id AND DeckQuantity>0 LIMIT 1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, b.Name, a.DeckQuantity AS
quantity FROM player_card a INNER JOIN cards b ON
a.CardID=b.CardID     WHERE     a.PlayerID=player_id     AND
a.DeckQuantity>0;
    ELSE
        SELECT 0 AS result, CONCAT("Empty Data") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.19. ***Stored Procedure*** **`get_player_deck`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_player_quest`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_player_quest`(player_id VARCHAR(50))
BEGIN
    DECLARE is_empty INT;
    SELECT  1  INTO  is_empty  FROM  player_quest  WHERE
PlayerID=player_id LIMIT 1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, BattleID, IsClear, IsActive FROM
player_quest WHERE PlayerID=player_id;
    ELSE
        SELECT 0 AS result, CONCAT("Empty Data") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.20. ***Stored Procedure*** **`get_player_quest`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_player_trunk`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_player_trunk`(player_id VARCHAR(50))
BEGIN
    DECLARE is_empty INT;
    SELECT  1  INTO  is_empty  FROM  player_card  WHERE
PlayerID=player_id AND TrunkQuantity>0 LIMIT 1;
    IF(is_empty!="") THEN
        SELECT  1  AS  result,  b.Name,  a.TrunkQuantity  AS
quantity  FROM  player_card  a  INNER  JOIN  cards  b  ON
a.CardID=b.CardID     WHERE     a.PlayerID=player_id     AND
a.TrunkQuantity>0;
    ELSE
        SELECT 0 AS result, CONCAT("Empty Data") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.21. ***Stored Procedure*** **`get_player_trunk`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_profile`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`get_profile`(player_id VARCHAR(50))
```

```
BEGIN
    DECLARE is_valid INT;
    SELECT 1 INTO is_valid FROM players WHERE
PlayerID=player_id LIMIT 1;
    IF(is_valid!="") THEN
        SELECT 1 AS result, p.PlayerID, p.Email, p.Gold,
            p.MaxHP, p.MaxDP, p.Gender,
            p.MaxSP, p.XP, p.Battle_Won, p.Battle_Lost,
            p.Energy, p.Job, p.Rank, p.Level
        FROM players p WHERE p.PlayerID=player_id;
    ELSE
        SELECT 0 AS result, CONCAT("Player is not
registered") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.22. ***Stored Procedure*** **`get_profile`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `get_trade_request_list`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`get_trade_request_list`(player_name varchar(50))
BEGIN
    DECLARE is_empty VARCHAR(1);
    SELECT 1 INTO is_empty FROM trading_detail WHERE
RequestedPlayer=player_name LIMIT 1;
    IF(is_empty!="") THEN
        SELECT 1 AS result, RequestID, SenderPlayer AS
PlayerID FROM trading_detail WHERE
RequestedPlayer=player_name;
    ELSE
        SELECT 0 AS result, CONCAT("No Incoming Trade
Request") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.23. ***Stored Procedure***
`get_trade_request_list`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `ignore_friend_request`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`ignore_friend_request`(player_one VARCHAR(50), player_two
```

```
VARCHAR(50))
BEGIN
    DECLARE is_valid VARCHAR(1);
    SELECT 1 INTO is_valid FROM player_request
    WHERE SenderPlayer=player_two AND
RequestedPlayer=player_one AND Information='friend request
sent';
    IF(is_valid!="") THEN
        DELETE FROM player_request
        WHERE SenderPlayer=player_two AND
RequestedPlayer=player_one AND Information='friend request
sent';
        SELECT 1 AS result, CONCAT("Friend Request Ignored")
AS info;
    ELSE
        SELECT 0 AS result, CONCAT("No Friend Request
Found") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.24. ***Stored Procedure*** **`ignore_friend_request`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `insert_card_request`$$
CREATE DEFINER=`yowanda1_cws`@`%` PROCEDURE
`insert_card_request`(player_id VARCHAR(50), request_id INT,
card_name VARCHAR(50), card_quantity INT)
BEGIN

    DECLARE card_id VARCHAR(50);

    SELECT c.CardID INTO card_id FROM cards c WHERE
c.Name=card_name;

    IF(card_id!="") THEN

        INSERT INTO trade_request_detail
VALUES('',player_id, request_id, card_id, card_quantity);

        SELECT 1 AS result, CONCAT("Trade Request Sent") AS
info;
    ELSE

        SELECT 0 AS result, CONCAT("Invalid card name") AS
info;
```

```
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.25. ***Stored Procedure*** **`insert_card_request`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `insert_to_deck`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`insert_to_deck`(player_id       VARCHAR(50),       card_name
VARCHAR(50), quantity INT)
BEGIN
    DECLARE card_id VARCHAR(50);
    SELECT  c.CardID  INTO  card_id  FROM  cards  c    WHERE
c.Name=card_name;
    IF(card_id!="") THEN
        UPDATE player_card SET TrunkQuantity=TotalQuantity-
quantity WHERE PlayerID=player_id AND CardID=card_id;
        UPDATE player_card SET DeckQuantity=quantity WHERE
PlayerID=player_id AND CardID=card_id;
        SELECT 1 AS result, CONCAT("Card has been added to
deck") AS info;
    ELSE
        SELECT 0 AS result, CONCAT("Invalid Card Name") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.26. ***Stored Procedure*** **`insert_to_deck`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `login`$$
CREATE       DEFINER=`yowanda1_cws`@`%`       PROCEDURE
`login`(p_username VARCHAR(50), p_pass VARCHAR(25))
BEGIN
    DECLARE login_status VARCHAR(1);
    SELECT 1 INTO login_status FROM players p WHERE
p.PlayerID=p_username AND p.Password=p_pass;
    IF(login_status != "") THEN
        SELECT 1 AS result, CONCAT("Login Succesful")
AS info;
    ELSE
        SELECT  0  AS  result,  CONCAT("Login  Failed.
```

```
Incorrect Username or Password") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.27. ***Stored Procedure*** **`login`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `receive_card`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`receive_card`(player_id VARCHAR(50), card_name VARCHAR(50),
quantity INT)
BEGIN
    DECLARE is_on_deck VARCHAR(1);
    DECLARE card_id VARCHAR(10);
    DECLARE card_price INT;
    DECLARE player_gold INT;
    SELECT c.CardID INTO card_id FROM cards c WHERE c.Name =
card_name LIMIT 1;
    SELECT ShopPrice INTO card_price FROM card_shop WHERE
CardID=card_id;
    SELECT Gold INTO player_gold FROM players WHERE
PlayerID=player_id;
    IF(quantity<=0) THEN
        SELECT -1 AS result, CONCAT("Invalid Card Quantity")
AS info;
    ELSE
        IF(card_price*quantity > player_gold) THEN
            SELECT -2 AS result, CONCAT("Not Enough Money")
AS info;
        ELSE
            IF(card_id!="") THEN
                SELECT 1 INTO is_on_deck FROM player_card
WHERE PlayerID=player_id AND CardID=card_id;
                IF(is_on_deck!="") THEN
                    UPDATE          player_card          SET
TotalQuantity=TotalQuantity+quantity,
TrunkQuantity=TrunkQuantity+quantity
                    WHERE       PlayerID=player_id       AND
CardID=card_id;
                ELSE
                    INSERT   INTO   player_card   VALUES('',
player_id, card_id, quantity, 0, quantity);
                END IF;
                UPDATE   players   p   SET   p.Gold=p.Gold-
(card_price*quantity);
                SELECT 1 AS result, CONCAT("Item Received")
AS info;
```

```
            ELSE
                SELECT 0 AS result, CONCAT("Invalid Card
Name") AS info;
            END IF;
        END IF;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.28. ***Stored Procedure*** **`receive_card`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `register`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`register`(p_username VARCHAR(50), p_fb VARCHAR(50), p_email
VARCHAR(25), p_job VARCHAR(15), p_pass VARCHAR(25))
BEGIN
    DECLARE username_isvalid VARCHAR(1);
    DECLARE is_registered VARCHAR(1);
    SELECT 1 INTO username_isvalid FROM players p WHERE
p.PlayerID=p_username;
    IF(p_fb!="") THEN
        SELECT 1 INTO is_registered FROM players p WHERE
p.FB_ID=p_fb LIMIT 1;
    END IF;
    IF(is_registered!="") THEN
        SELECT -1 AS result, CONCAT("Registration Failed.
Facebook is already registered") AS info;
    ELSE
        IF(username_isvalid!="") THEN
            SELECT 0 AS result, CONCAT("Username is not
valid or has been taken") AS info;
        ELSE
            INSERT INTO players VALUES(p_username, p_fb,
p_pass,                                                 p_email,
100,100,40,3,0,0,0,10,p_job,'D',1,'Female');
            SELECT 1 AS result, CONCAT("Registration
Success") AS info;
        END IF;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.29. ***Stored Procedure*** **`register`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `remove_card`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
```

```
`remove_card`(player_id VARCHAR(50), card_name VARCHAR(50),
quantity INT)
BEGIN
    DECLARE is_on_deck VARCHAR(1);
    DECLARE sell_price INT;
    DECLARE total INT;
    DECLARE total_in_deck INT;
    DECLARE card_id VARCHAR(10);
    SELECT c.CardID INTO card_id FROM cards c WHERE c.Name =
card_name LIMIT 1;
    SELECT 1 INTO is_on_deck FROM player_card WHERE
PlayerID=player_id AND CardID=card_id;
    SELECT c.SellPrice INTO sell_price FROM cards c WHERE
c.CardID=card_id;
    IF(is_on_deck!="") THEN
        SELECT TotalQuantity INTO total FROM player_card
WHERE PlayerID=player_id AND CardID=card_id;
        IF(total<quantity || quantity<=0) THEN
            SELECT -1 as result, CONCAT("Invalid Card
Quantity") as info;
        ELSE
            SELECT DeckQuantity INTO total_in_deck FROM
player_card WHERE PlayerID=player_id AND CardID=card_id;

            IF(total-quantity < total_in_deck) THEN
                SELECT -1 as result, CONCAT("Invalid Number
of Card. Remove From Deck First") as info;
            ELSE
                IF(total-quantity=0) THEN

                    DELETE FROM player_card WHERE
PlayerID=player_id AND CardID=card_id;
                ELSE
                    UPDATE player_card SET
TotalQuantity=TotalQuantity-quantity,
TrunkQuantity=TrunkQuantity-quantity
                    WHERE PlayerID=player_id AND
CardID=card_id;
                END IF;
                UPDATE players p SET
p.Gold=p.Gold+(sell_price*quantity);
                SELECT 1 AS result, CONCAT("Card Removed")
AS info;
            END IF;
        END IF;
    ELSE

        SELECT 0 AS result, CONCAT("Invalid Card Name") AS
info;
```

```
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.30. ***Stored Procedure*** **`remove_card`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `remove_friend`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`remove_friend`(player VARCHAR(50), friend_name VARCHAR(50))
BEGIN
    DECLARE is_on_party VARCHAR(1);
    DECLARE is_friend VARCHAR(1);
    SELECT  1  INTO  is_friend  FROM  player_friend  WHERE
PlayerOne=player AND PlayerTwo=friend_name;
    IF(is_friend!="") THEN
        DELETE FROM player_friend WHERE PlayerOne=player AND
PlayerTwo=friend_name;
        DELETE        FROM        player_friend        WHERE
PlayerOne=friend_name AND PlayerTwo=player;
        SELECT 1 AS result, CONCAT(friend_name, " has been
removed from your friend list") AS info;
    ELSE
        SELECT 0 AS result, CONCAT(friend_name, " is not in
your friend list") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.31. ***Stored Procedure*** **`remove_friend`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `send_battle_result`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`send_battle_result`(player_win   VARCHAR(50),   player_lost
VARCHAR(50))
BEGIN
    DECLARE had_battle INT;
    DECLARE win_points INT;
    DECLARE lost_points INT;
    SELECT  Battle_Won  INTO  win_points  FROM  players  WHERE
PlayerID=player_win;
    SELECT Battle_Lost INTO lost_points FROM players WHERE
PlayerID=player_lost;
    UPDATE   players   SET   Battle_Won=win_points+1   WHERE
PlayerID=player_win;
    UPDATE   players   SET   Battle_Lost=lost_points+1   WHERE
PlayerID=player_lost;
```

```
    SELECT 1 INTO had_battle FROM battle_detail WHERE
PlayerOne=player_win AND PlayerTwo=player_lost;
    IF(had_battle!="") THEN
        SELECT Win INTO win_points FROM battle_detail WHERE
PlayerOne=player_win AND PlayerTwo=player_lost;
        SELECT Lose INTO lost_points FROM battle_detail
WHERE PlayerOne=player_lost AND PlayerTwo=player_win;
        UPDATE battle_detail SET Win=win_points+1 WHERE
PlayerOne=player_win AND PlayerTwo=player_lost;
        UPDATE battle_detail SET Lose=lost_points+1 WHERE
PlayerOne=player_lost AND PlayerTwo=player_win;
    ELSE
        INSERT INTO battle_detail VALUES('',player_win,
player_lost,1,0);
        INSERT INTO battle_detail VALUES('',player_lost,
player_win,0,1);
    END IF;
    SELECT 1 AS result, CONCAT("Battle Result Updated") AS
info;
    END$$
DELIMITER ;
```

Kode Sumber 7.32. ***Stored Procedure*** **`send_battle_result`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `send_energy_request`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`send_energy_request`(sender_player              VARCHAR(50),
requested_player VARCHAR(50))
BEGIN
    DECLARE start_time DATETIME;
    DECLARE end_time DATETIME;
    DECLARE time_dif INT;
    DECLARE is_valid VARCHAR(1);
    SELECT RequestTime INTO start_time FROM player_request
    WHERE           SenderPlayer=sender_player           AND
RequestedPlayer=requested_player
    AND Information LIKE '%nergy%'
    ORDER BY RequestTime DESC LIMIT 1;
    SELECT NOW() INTO end_time;
    SELECT DATEDIFF(end_time, start_time) INTO time_dif;
    SELECT    1    INTO    is_valid    FROM    players    WHERE
PlayerID=requested_player;
    IF(is_valid!="") THEN
        IF(time_dif<1) THEN
            SELECT -1 AS result, CONCAT("You can only send
extra energy once per day for each friend") AS info;
```

```
        ELSE
            INSERT            INTO            player_request
VALUES('',sender_player,requested_player,NOW(),'Energy
Sent');
            SELECT  1  AS  result,  CONCAT("Energy  Sent")  AS
info;
        END IF;
    ELSE
        SELECT 0 AS result, CONCAT("Invalid Player Name") AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.33. ***Stored Procedure*** **`send_energy_request`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `send_friend_request`$$
CREATE       DEFINER=`yowanda1_cws`@`%`       PROCEDURE
`send_friend_request`(player_one           VARCHAR(50),
player_two VARCHAR(50))
BEGIN
    DECLARE is_requested VARCHAR(1);
    DECLARE is_valid VARCHAR(1);
    DECLARE is_friend VARCHAR(1);
    SELECT 1 INTO is_requested FROM player_request
    WHERE          SenderPlayer=player_one          AND
RequestedPlayer=player_two   AND   Information='friend
request sent';
    SELECT  1  INTO  is_valid  FROM  players  WHERE
PlayerID=player_two;
    IF(is_valid!="") THEN
        IF(is_requested!="") THEN
            SELECT 0 AS result, CONCAT(player_two," is
already on your friend request list") AS info;
        ELSE
            SELECT 1 INTO is_friend FROM player_friend
WHERE PlayerOne=player_one AND PlayerTwo=player_two;
            IF(is_friend!="") THEN
                SELECT       -1       AS       result,
CONCAT(player_two," is already on your friend list")
AS info;
            ELSE
                INSERT       INTO       player_request
```

```
VALUES('',player_one,player_two,NOW(),'Friend  Request
Sent');
                SELECT  1  AS  result,  CONCAT("Friend
Request Sent") AS info;
            END IF;
        END IF;
    ELSE
        SELECT  -2  AS  result,  CONCAT("Invalid  Player
Name") AS info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.34. ***Stored Procedure*** **`send_friend_request`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `send_message`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`send_message`(player_one       VARCHAR(50),       player_two
VARCHAR(50), msg_subject TEXT, msg TEXT)
BEGIN
    INSERT    INTO    player_message    VALUES('',player_one,
player_two, msg_subject, msg);
    SELECT 1 AS result, CONCAT("Message Sent") AS info;
    END$$
DELIMITER ;
```

Kode Sumber 7.35. ***Stored Procedure*** **`send_message`**

```
DELIMITER $$

USE `yowanda1_cws`$$

DROP PROCEDURE IF EXISTS `send_trade_request`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`send_trade_request`(player_one    VARCHAR(50),    player_two
VARCHAR(50))

BEGIN

    INSERT    INTO    trading_detail    VALUES('',player_one,
player_two);

    SELECT r.RequestID FROM trading_detail r
    WHERE           r.SenderPlayer=player_one           AND
r.RequestedPlayer=player_two
```

```
    ORDER BY r.RequestID DESC LIMIT 1;
    END$$
DELIMITER ;
```

Kode Sumber 7.36. ***Stored Procedure*** **`send_trade_request`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `show_battle_rank`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`show_battle_rank`(player_id VARCHAR(50))
BEGIN
    DECLARE is_valid VARCHAR(1);
    SELECT  1  INTO  is_valid  FROM  battle_detail  WHERE
PlayerOne=player_id LIMIT 1;
    IF(is_valid!="") THEN
        SELECT 1 AS result, PlayerTwo AS Enemy, Win, Lose
FROM battle_detail WHERE PlayerOne=player_id;
    ELSE
        SELECT  0  AS  result,  CONCAT("Data  Not  Found")  AS
info;
    END IF;
    END$$
DELIMITER ;
```

Kode Sumber 7.37. ***Stored Procedure*** **`show_battle_rank`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `show_player_ranking`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`show_player_ranking`()
BEGIN
    SELECT PlayerID AS player, Battle_Won AS points FROM
players ORDER BY Battle_Won DESC LIMIT 5;
    END$$
DELIMITER ;
```

Kode Sumber 7.38. ***Stored Procedure*** **`show_player_ranking`**

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `update_building`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`update_building`(player_id VARCHAR(50), b_name VARCHAR(15),
b_level INT,
    b_maxXP INT, b_currentXP INT, b_quantity INT, b_claimed
```

```
INT, b_prefabName VARCHAR(50))
BEGIN
    UPDATE player_building B SET
    B.Level=b_level,                          B.maxXP=b_maxXP,
B.CurrentXP=b_currentXP,
    B.ProductionQuantity=b_quantity,
B.ProductClaimed=b_claimed,
    B.PrefabName=b_prefabName
    WHERE B.PlayerID=player_id AND B.Name=b_name;
    SELECT 1 AS result, CONCAT("Building Updated") AS info;
    END$$ DELIMITER ;
```

Kode Sumber 7.39. *Stored Procedure* `update_building`

```
DELIMITER $$
USE `yowanda1_cws`$$
DROP PROCEDURE IF EXISTS `update_player_data`$$
CREATE          DEFINER=`yowanda1_cws`@`%`          PROCEDURE
`update_player_data`(player_id VARCHAR(50), gold INT, max_hp
INT,
    max_dp INT, max_sp INT, xp INT, energy INT,
    job VARCHAR(15), rank VARCHAR(1), player_level INT)
BEGIN
    UPDATE players p SET
    p.Gold=gold, p.Xp=xp, p.Energy=energy,
    p.MaxHP=max_hp, p.MaxDP=max_dp, p.MaxSP=max_sp,
    p.Job=job, p.Rank=rank, p.Level=player_level
    WHERE p.PlayerID=player_id;
    SELECT 1 AS result, CONCAT("Data Updated") AS info;
    END$$
DELIMITER ;
```

Kode Sumber 7.40. *Stored Procedure* `update_player_data`

```
<?php
    class avatar extends CI_Model
    {


        function get_avatar($function_name, $player_name)
        {

            //Calls Stored Procedure
            $stored_procedure            =            "CALL
get_".$function_name."_avatar(?)";
            $param = array($player_name);
            $query    =    $this->db->query($stored_procedure,
$param);
```

```
            //Creates XML Document
            $xml = new DOMDocument("1.0");
            $data                    =                    $xml-
>createElement("AvatarFromService");
            $xml->appendChild($data);



            foreach($query->result() as $value)
            {

                $slot = $xml->createElement("Slot");
                $slot_text   =   $xml->createTextNode($value-
>Slot);
                $slot->appendChild($slot_text);

                $name = $xml->createElement("Name");
                $name_text   =   $xml->createTextNode($value-
>Name);
                $name->appendChild($name_text);

                $avatar                =                $xml-
>createElement("AviFromService");
                $avatar->appendChild($slot);
                $avatar->appendChild($name);
                $data->appendChild($avatar);
            }
            $xml->formatOutput = true;
            $xml-
>save("files/".$function_name."_avatar_of_".$player_name.".x
ml");
            return   "1|XML   File   Has   Been   Successfully
Generated";
        }



        function edit_avatar($player_name, $avatar)
        {

            list($head,   $eye,   $mouth)   =   explode("-",
$avatar);
            //Calls Stored Procedure
            $stored_procedure = "CALL edit_avatar(?,?,?,?)";
            $param   =   array($player_name,   $head,   $eye,
$mouth);
            $query   =   $this->db->query($stored_procedure,
$param);
```

```
            foreach($query->result()    as    $value)    return
$value->result . "|" . $value->info;
        }
    }
?>
```

Kode Sumber 7.41. Implementasi Kelas Data `avatar`

```
<?php
    class battle extends CI_Model
    {
        function              send_battle_result($player_name,
$enemy_name)
        {
            //Calls Stored Procedure
            $stored_procedure              =              "CALL
send_battle_result(?,?)";
            $param = array($player_name, $enemy_name);
            $query    =    $this->db->query($stored_procedure,
$param);
            foreach($query->result()    as    $value)    return
$value->result . "|" . $value->info;
        }

        function show_player_ranking()
        {
            //Calls Stored Procedure
            $stored_procedure              =              "Call
show_player_ranking()";
            $query = $this->db->query($stored_procedure);

            //Creates XML Document
            $xml = new DOMDocument("1.0");
            $data                          =                    $xml-
>createElement("PlayerRankingFromService");
            $xml->appendChild($data);
            foreach($query->result() as $value)
            {
                $name = $xml->createElement("Name");
                $name_text    =    $xml->createTextNode($value-
>player);
                $name->appendChild($name_text);

                $points = $xml->createElement("BattleWon");
                $points_text  =  $xml->createTextNode($value-
>points);
                $points->appendChild($points_text);
```

```
                $player                 =                 $xml-
>createElement("PlayerStatsFromService");
                $player->appendChild($name);
                $player->appendChild($points);
                $data->appendChild($player);
            }
            $xml->formatOutput = true;
            $xml->save("files/player_rank.xml");
            return   "1|XML   File   Has   Been   Successfully
Generated";
        }

        function show_battle_rank($player_name)
        {
            //Calls Stored Procedure
            $stored_procedure = "CALL show_battle_rank(?)";
            $param = array($player_name);
            $query    =    $this->db->query($stored_procedure,
$param);

            //Creates XML Document
            $xml = new DOMDocument("1.0");
            $data                       =                 $xml-
>createElement("BattleRankFromService");
            $xml->appendChild($data);
            foreach($query->result() as $value)
            {
                if($value->result!="1")     return     $value-
>result . "|" . $value->info;

                $name = $xml->createElement("Name");
                $name_text    =    $xml->createTextNode($value-
>Enemy);
                $name->appendChild($name_text);

                $win = $xml->createElement("Win");
                $win_text    =    $xml->createTextNode($value-
>Win);
                $win->appendChild($win_text);

                $lose = $xml->createElement("Lose");
                $lose_text    =    $xml->createTextNode($value-
>Lose);
                $lose->appendChild($lose_text);

                $enemy                  =                 $xml-
>createElement("EnemyPlayerFromService");
                $enemy->appendChild($name);
                $enemy->appendChild($win);
```

```
                $enemy->appendChild($lose);
                $data->appendChild($enemy);
            }
            $xml->formatOutput = true;
            $xml-
>save("files/battle_rank_of_".$player_name.".xml");
            return   "1|XML   File   Has   Been   Successfully
Generated";
        }

        function   calculate_data($player_name,   $xp_earned,
$gold_earned)
        {
            $this->db->trans_start();
            //Calls Stored Procedure
            $stored_procedure = "CALL get_profile(?)";
            $param = array($player_name);
            $query   =   $this->db->query($stored_procedure,
$param);

            $current_rank = '';
            $current_level = 0;
            $current_job = '';
            $current_gold = 0;
            $xp_pool = 0;
            $current_xp = 0;
            foreach($query->result() as $value)
            {
                $xp_pool  =  $this->get_xp_pool($value->Rank,
$value->Level);
                $current_level = $value->Level;
                $current_rank = $value->Rank;
                $current_job = $value->Job;
                $current_xp = $value->XP;
                $current_gold = $value->Gold + $gold_earned;
            }

            $lvl_up_counter=0;
            $current_xp += $xp_earned;
            while($current_xp >= $xp_pool)
            {
                $current_xp -= $xp_pool;
                $current_level++;
                if($current_level>20)
                {
                    $current_level -= 20;
                    $current_rank          =          $this-
>level_up($current_rank);
                }
```

```
                $xp_pool = $this->get_xp_pool($current_rank,
$current_level);
                $lvl_up_counter++;
            }

            if($lvl_up_counter>0)
            {
                return "1|Level Up! You are now
".$current_job." Rank ".$current_rank." Level
".$current_level.".";
            }
            return "1|Gold and XP Updated";
        }

        function level_up($player_rank)
        {
            $data['rank'] = array(
              1 => "D",
              2 => "C",
              3 => "B",
              4 => "A",
              5 => "S"
            );

            $input = array_search($player_rank,
$data['rank']);
            $input++;
            return $data['rank'][$input];
        }

        function get_xp_pool($player_rank, $player_level)
        {
            $data['counter'] = 0;

            $data['rank_index'] = array(
              "D" => 1,
              "C" => 1.25,
              "B" => 1.5,
              "A" => 1.75,
              "S" => 2
            );

            $data['rank'] = array(
              1 => "D",
              2 => "C",
              3 => "B",
              4 => "A",
              5 => "S"
            );
```

```
            $input        =        (array_search($player_rank,
$data['rank']) * $player_level) - 1;

            $rank = 'D';
            $xp_pool = 0;
            $rank_to_index = 1;
            $new_xp_pool=100;
            $data['xp_pool']=array();

            for($i=1;$i<=5;$i++)
            {
                for($j=1;$j<=20;$j++)
                {
                    $value                                          =
floor($data['rank_index'][$data['rank'][$i]] * ceil($j/4));
                    $new_xp_pool   =   ($xp_pool   *   1.5)   +
($value * 50);
                    $xp_mod_100 = $new_xp_pool%100;
                    $xp_per_100 = floor($new_xp_pool/100);

                    if($xp_mod_100 >0)
                    {
                        $new_xp_pool = ($xp_per_100 * 100) +
100;
                    }
                    $data['xp_pool'][$data['counter']]      =
$new_xp_pool;
                    if($j%4==0)
                    {
                        $xp_pool = $new_xp_pool;
                    }
                    $rank = $data['rank'][$rank_to_index];
                    $data['counter']++;
                }
                $xp_pool = 0;
            }
            return $data['xp_pool'][$input];
        }
    }
?>
```

Kode Sumber 7.42. Implementasi Kelas Data `battle`

```
<?php
    class building extends CI_Model
    {
        function get_building($player_name)
        {
```

```
            //Calls Stored Procedure
            $stored_procedure              =              "CALL
get_player_building(?)";
            $param = array($player_name);
            $query    =    $this->db->query($stored_procedure,
$param);

            //Creates XML Document
            $xml = new DOMDocument("1.0");
            $data                    =                    $xml-
>createElement("PlayerBuildingFromService");
            $xml->appendChild($data);

            foreach($query->result() as $value)
            {
                $name = $xml->createElement("Name");
                $name_text   =   $xml->createTextNode($value-
>Name);
                $name->appendChild($name_text);

                $level = $xml->createElement("Level");
                $level_text   =   $xml->createTextNode($value-
>Level);
                $level->appendChild($level_text);

                $max_xp = $xml->createElement("MaxXP");
                $max_xp_text  =  $xml->createTextNode($value-
>MaxXP);
                $max_xp->appendChild($max_xp_text);

                $current_xp               =               $xml-
>createElement("CurrentXP");
                $current_xp_text            =            $xml-
>createTextNode($value->CurrentXP);
                $current_xp->appendChild($current_xp_text);

                $quantity                 =                $xml-
>createElement("ProductionQuantity");
                $quantity_text             =             $xml-
>createTextNode($value->ProductionQuantity);
                $quantity->appendChild($quantity_text);

                $claimed                  =                $xml-
>createElement("ProductClaimed");
                $claimed_text  =  $xml->createTextNode($value-
>ProductClaimed);
                $claimed->appendChild($claimed_text);

                $prefab_name              =              $xml-
```

```
>createElement("PrefabName");
                $prefab_name_text           =           $xml-
>createTextNode($value->MaxXP);
                $prefab_name-
>appendChild($prefab_name_text);

                $building               =               $xml-
>createElement("BuildingFromService");
                $building->appendChild($name);
                $building->appendChild($level);
                $building->appendChild($max_xp);
                $building->appendChild($current_xp);
                $building->appendChild($quantity);
                $building->appendChild($claimed);
                $building->appendChild($prefab_name);
                $data->appendChild($building);
            }

            $xml->formatOutput = true;


$xml->save("files/building_of_".$player_name.".xml");

 return "1|XML File Has Been Successfully Generated";
        }


function    update_building($building_stats,    $level_stats,
$product_stats)
        {

            list($id,  $name,  $prefabName)  =  explode("-",
$building_stats);
            list($level,  $maxXP,  $currentXP)  =  explode("-",
$level_stats);
            list($productionQuantity,    $productClaimed)    =
explode("-", $product_stats);
            //Calls Stored Procedure
            $stored_procedure             =             "CALL
update_building(?,?,?,?,?,?,?,?)";
            $param  =  array($id,  $name,  $level,  $maxXP,
$currentXP,
                $productionQuantity,         $productClaimed,
$prefabName);
            $query   =   $this->db->query($stored_procedure,
$param);

foreach($query->result() as $value) return $value->result .
"|" . $value->info;
```

```
        }
    }
?>
```

Kode Sumber 7.43. Implementasi Kelas Data `building`

```
<?php
    class card extends CI_Model
    {
        function   insert_to_deck($player_name,   $card_name,
$quantity)
        {
            //Calls Stored Procedure
            $stored_procedure             =             "CALL
insert_to_deck(?,?,?)";
            //echo $card name;
            $card_name    =    str_replace("%20",    "    ",
$card_name);
            $param    =    array($player_name,    $card_name,
$quantity);
            $query    =    $this->db->query($stored_procedure,
$param);
            foreach($query->result()    as    $value)    return
$value->result . "|" . $value->info;
            //return $card_name;
        }

        function get_cards($function_name, $player_name)
        {
            //Calls Stored Procedure
            $stored_procedure             =             "CALL
get_player_".$function_name."(?) ";
            $param = array($player_name);
            $query    =    $this->db->query($stored_procedure,
$param);

            //Creates XML Document
            $xml = new DOMDocument("1.0");
            $data = $xml->createElement("Data");
            $xml->appendChild($data);

            foreach($query->result() as $value)
            {
                if($value->result!="1")     return     $value-
>result . "|" . $value->info;

                $name = $xml->createElement("Name");
                $name_text              =              $xml-
>createTextNode(str_replace("%20", " ", $value->Name));
                $name->appendChild($name_text);
```

```
                $quantity = $xml->createElement("Quantity");
                $quantity_text              =              $xml-
>createTextNode($value->quantity);
                $quantity->appendChild($quantity_text);

                $cards = $xml->createElement("Card");
                $cards->appendChild($name);
                $cards->appendChild($quantity);
                $data->appendChild($cards);
            }
            $xml->formatOutput = true;
            $xml-
>save("files/".$function_name."_of_".$player_name.".xml");
            return   "1|XML   File   Has   Been   Successfully
Generated";
        }

        function clear_deck($player_name)
        {
            //Calls Stored Procedure
            $stored_procedure = "CALL clear_deck(?)";
            $param = array($player_name);
            $query    =    $this->db->query($stored_procedure,
$param);
            foreach   ($query->result()   as   $value)   return
$value->result . "|" . $value->info;
        }

        function      buy_card($player_name,      $card_name,
$quantity)
        {
            //Calls Stored Procedure
            $stored_procedure = "CALL receive_card(?,?,?)";
            $card = str_replace("%20", " ", $card_name);
            $param = array($player_name, $card, $quantity);
            $query    =    $this->db->query($stored_procedure,
$param);
            foreach   ($query->result()   as   $value)   return
$value->result . "|" . $value->info;
        }

        function      sell_card($player_name,      $card_name,
$quantity)
        {
            //Calls Stored Procedure
            $stored_procedure = "CALL remove_card(?,?,?)";
            $card_name     =     str_replace("%20",     "     ",
$card_name);
```

```
            $param    =    array($player_name,    $card_name,
$quantity);
            $query    =    $this->db->query($stored_procedure,
$param);
            foreach   ($query->result()   as   $value)   return
$value->result . "|" . $value->info;
        }

        function get_shop_cards()
        {
            //Calls Stored Procedure
            $stored_procedure = "CALL get_shop_cards()";
            $param = array();
            $query    =    $this->db->query($stored_procedure,
$param);

            //Creates XML Document
            $xml = new DOMDocument("1.0");
            $data = $xml->createElement("Cards_on_Shop");
            $xml->appendChild($data);

            foreach($query->result() as $value)
            {
                $name = $xml->createElement("Name");
                $name_text   =   $xml->createTextNode($value-
>Name);
                $name->appendChild($name_text);

                $price = $xml->createElement("Price");
                $price_text   =   $xml->createTextNode($value-
>ShopPrice);
                $price->appendChild($price_text);

                $cards = $xml->createElement("Card");
                $cards->appendChild($name);
                $cards->appendChild($price);
                $data->appendChild($cards);
            }
            $xml->formatOutput = true;
            $xml->save("files/cards_on_shop.xml");
            return   "1|XML   File   Has   Been   Successfully
Generated";
        }
    }
?>
```

Kode Sumber 7.44. Implementasi Kelas Data `card`

```
function process_request($class, $method, $param)
        {
```

```
            $this->load->library("Nusoap_Library");
            $this->nusoap_client            =            new
nusoap_client('http://cws.yowanda.com/ServerController?wsdl'
);
            $err = $this->nusoap_client->getError();
            $data['result'] = "";
            if ($err)
            {
                $data['result'] = $err;

            }
            else
            {
                $result  =  $this->nusoap_client->call($class
.'.'.$method, $param);

 if ($this->nusoap_client->fault) $data['result'] = $result;
                else
                {
                    $err = $this->nusoap_client->getError();
                    if ($err) $data['result'] = $err;
                    else $data['result'] = $result;
                }
            }



            $this->load->view('ClientView', $data);
        }
```

Kode Sumber 7.45. Potongan Kode Implementasi Pemanggilan Fungsi pada Kelas Kontrol

```
function index()
        {
            //Load Model and Methods
            //1. Player (Player's Data and Friend)
            $this->load->model('player');
            //2. Request (Friend Request)
            $this->load->model('request');
            //3. Card (Card Data and Deck)
            $this->load->model('card');
            //4. Battle (History and Result)
            $this->load->model('battle');
            //5. Battle (Player Avatar)
            $this->load->model('avatar');
            //6. Building (Player Building)
            $this->load->model('building');
            //7. Player Message (Private Messages)
```

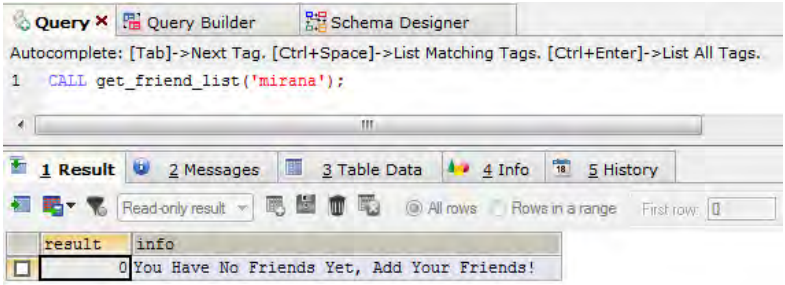
```
            $this->load->model('message');
            //8. Quest and Monster Data
            $this->load->model('monster_quest');
            $this->nusoap_server-
>service(file_get_contents("php://input"));
        }
```

Kode Sumber 7.46. Potongan Kode Implementasi *Load Model* pada Kelas Kontrol

LAMPIRAN B. HASIL PENGUJIAN

Tabel 8.1. Pengujian *Stored Procedure get_friend_list* Skenario 2

ID	UJ.UC-0002
Nama	Pengujian *stored procedure* menampilkan daftar teman yang dimiliki oleh seorang pemain.
Tujuan Pengujian	Menguji fungsi untuk menampilka daftar teman dari seorang pemain.
Skenario	Eksekusi *stored procedure* menghasilkan data yang kosong.
Langkah Pengujian	*Store procedure* dipanggil dengan menggunakan *parameter* ID pemain yang belum memiliki teman.
Hasil Yang Diharapkan	*Stored procedure* mengembalikan keterangan bahwa pemain belum memiliki teman.
Hasil Yang Didapat	Keterangan bahwa pemain belum memiliki teman.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 8.1

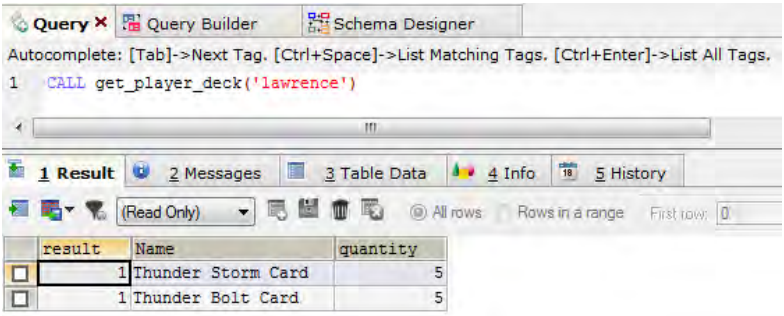


Gambar 8.1. Hasil Uji Skenario 2

Tabel 8.2. Pengujian *Stored Procedure get_player_deck* Skenario 3

ID	UJ.UC-0003
Nama	Pengujian *stored procedure* menampilkan daftar kartu yang dimiliki oleh seorang pemain.
Tujuan Pengujian	Menguji fungsi untuk menampilkan daftar kartu yang dimiliki dari seorang pemain yang ada di dalam *deck.*
Skenario	Eksekusi *stored procedure* menghasilkan satu atau lebih data.
Langkah Pengujian	*Store procedure* dipanggil dengan menggunakan *parameter* ID pemain yang belum memiliki teman.
Hasil Yang Diharapkan	*Stored procedure* mengembalikan data kartu yang dimiliki pemain yang ada pada *deck* pemain.
Hasil Yang Didapat	Data kartu yang ada pada *deck* pemain.
Hasil Pengujian	Berhasil

Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 8.2

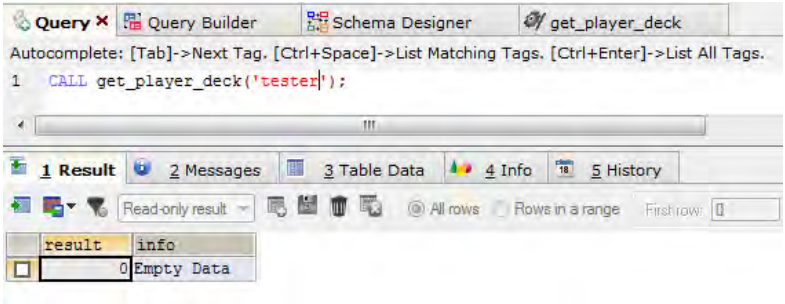


Gambar 8.2. Hasil Uji Skenario 3

Tabel 8.3. Pengujian *Stored Procedure get_player_deck*
Skenario 4

ID	UJ.UC-0004
Nama	Pengujian *stored procedure* menampilkan daftar kartu yang dimiliki oleh seorang pemain.
Tujuan Pengujian	Menguji fungsi untuk menampilkan daftar kartu yang dimiliki dari seorang pemain yang ada di dalam *deck*.
Skenario	Eksekusi *stored procedure* menghasilkan data kosong.
Langkah Pengujian	*Store procedure* dipanggil dengan menggunakan parameter ID pemain yang tidak mempunyai kartu pada *deck*.
Hasil Yang Diharapkan	*Stored procedure* mengembalikan keterangan bahwa data kosong.
Hasil Yang Didapat	Keterangan bahwa data hasil eksekusi *stored procedure* kosong.

Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 8.3.

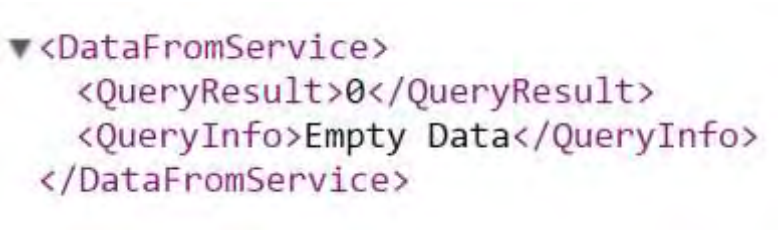


Gambar 8.3. Hasil Uji Skenario 4

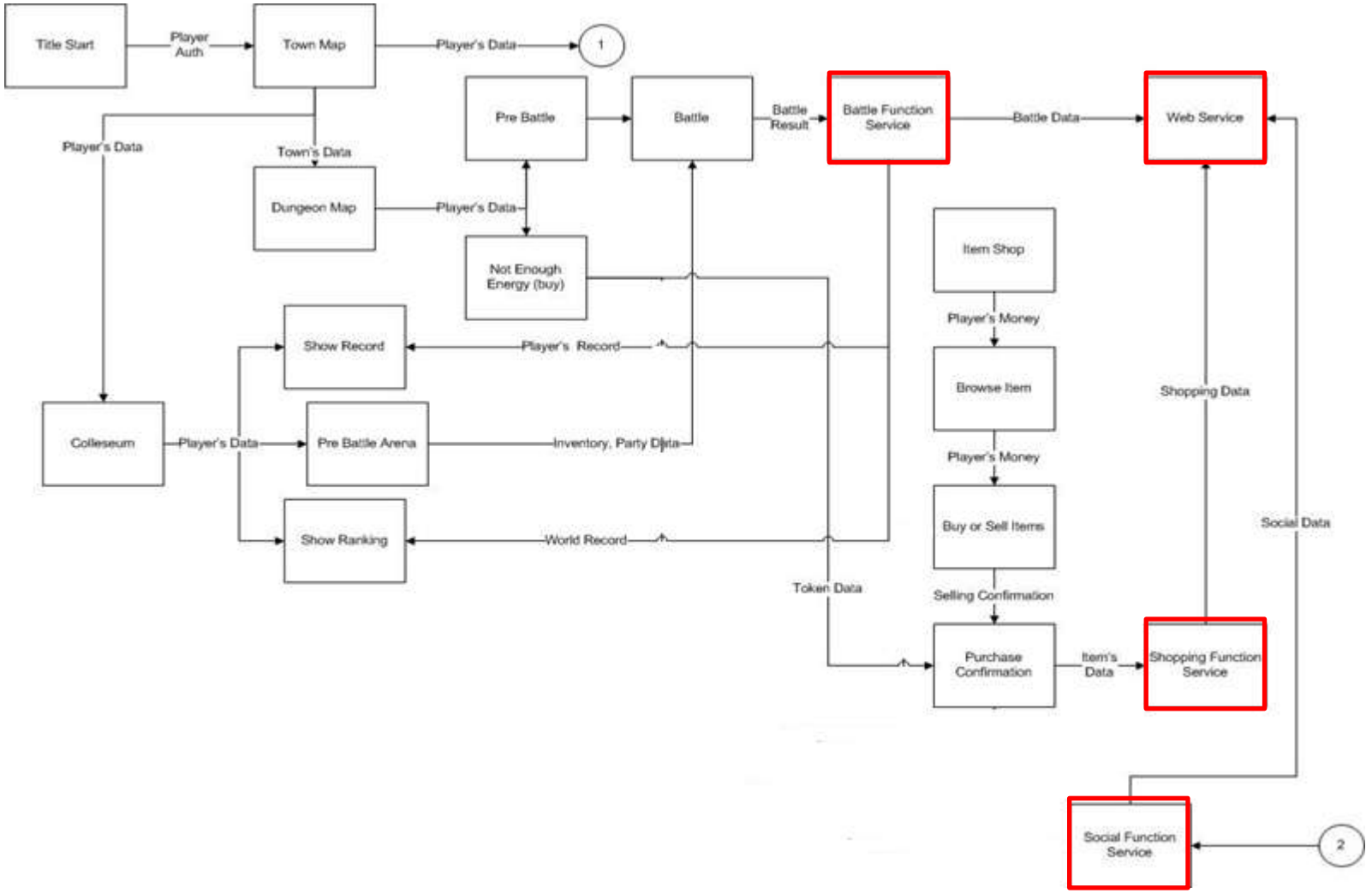
Tabel 8.4. Pengujian Fungsi *get_player_deck* Skenario 2

ID	UJ.UC-0012
Nama	Pengujian fungsi pada *web service* menampilkan daftar kartu yang dimiliki oleh seorang pemain.
Tujuan Pengujian	Menguji fungsi untuk menampilkan daftar kartu yang dimiliki dari seorang pemain yang ada di dalam *deck.*
Skenario	Eksekusi fungsi pada *web service* menghasilkan data kosong.
Langkah Pengujian	Fungsi pada *web service* dipanggil dengan menggunakan parameter ID pemain yang tidak mempunyai kartu pada *deck.*
Hasil Yang Diharapkan	Fungsi pada *web service* mengembalikan keterangan bahwa data kosong.
Hasil Yang Didapat	Keterangan bahwa data hasil eksekusi fungsi pada *web service* kosong.

Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil uji coba dapat dilihat pada Gambar 8.4.

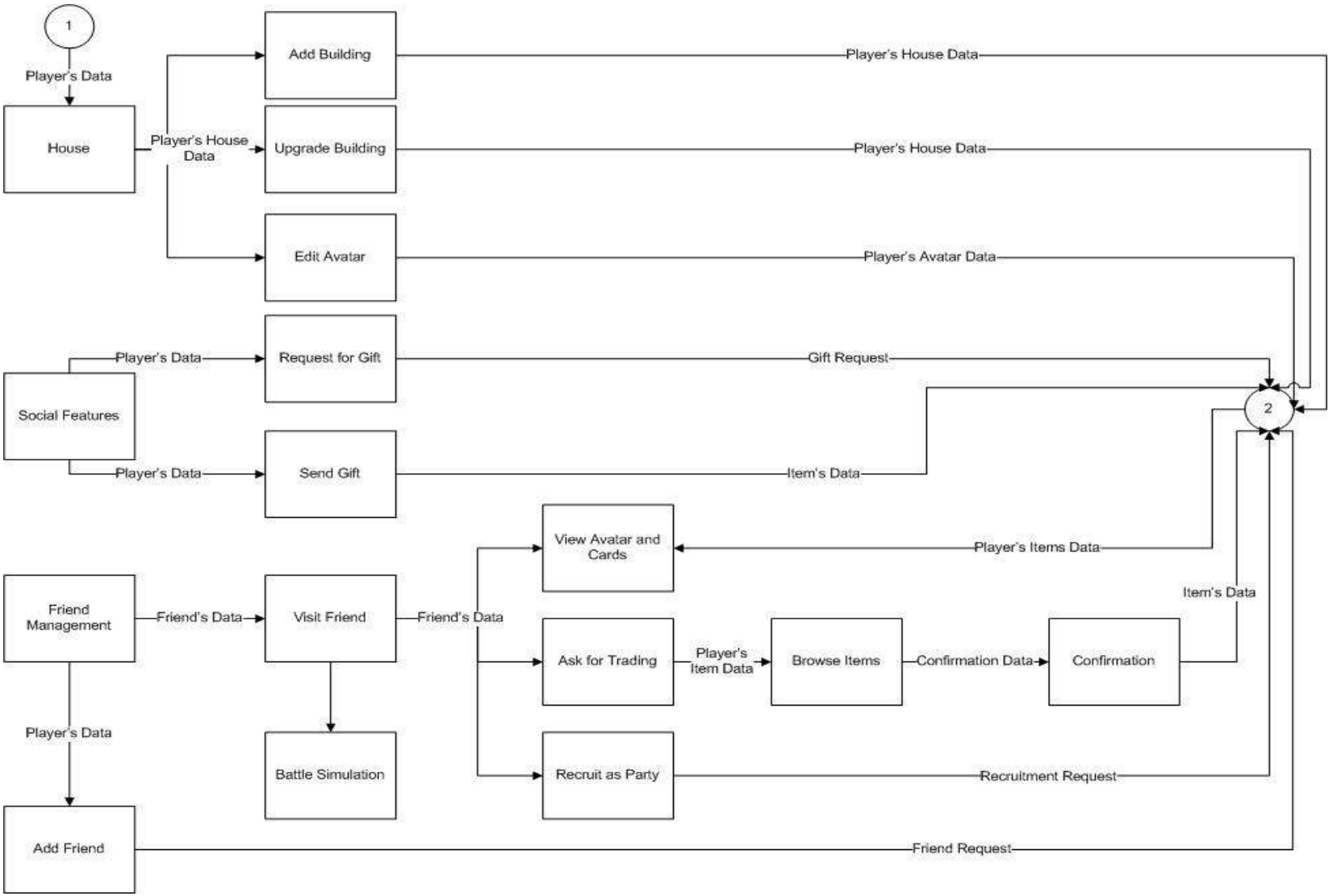


Gambar 8.4. Hasil Uji Skenario 2 Fungsi *Web Service*

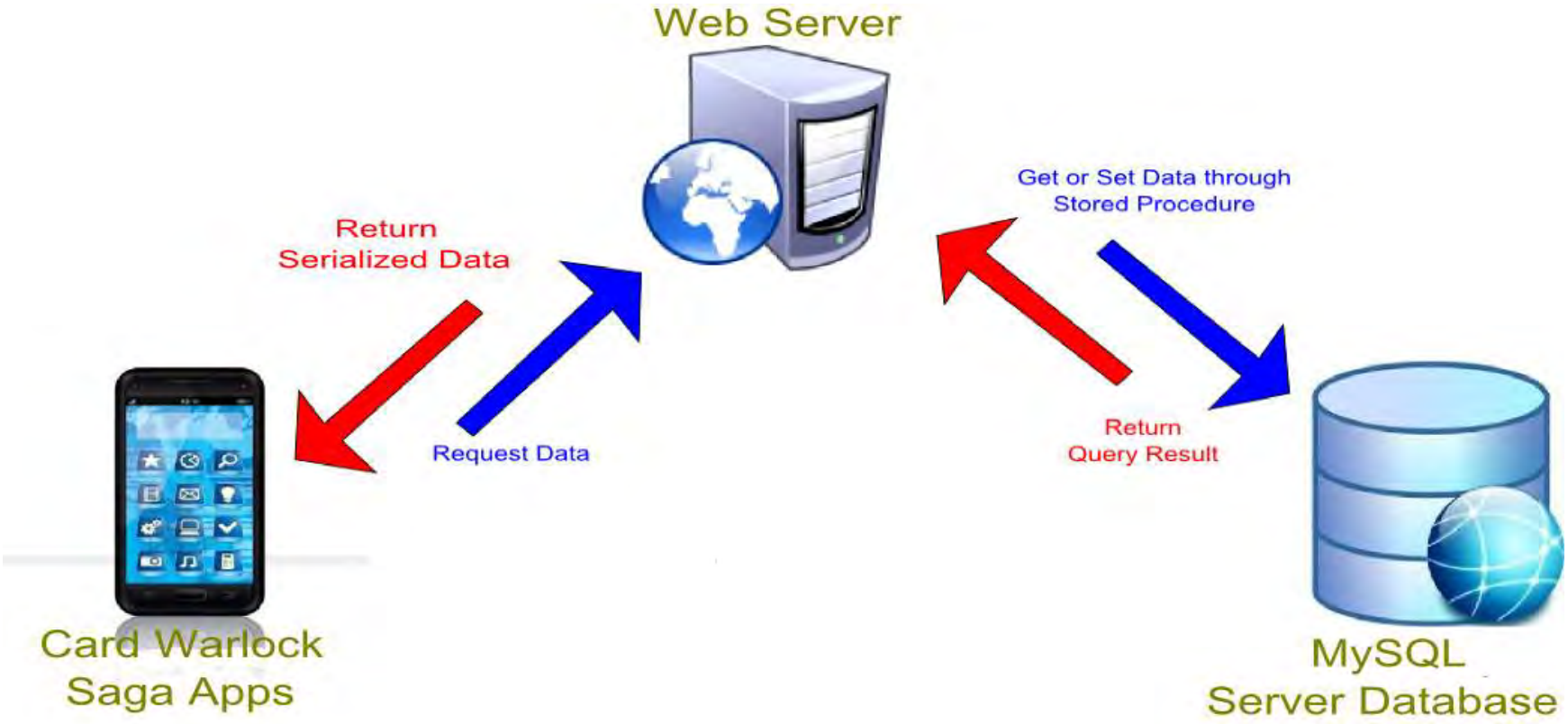


LAMPIRAN C. GAMBAR DAN DIAGRAM

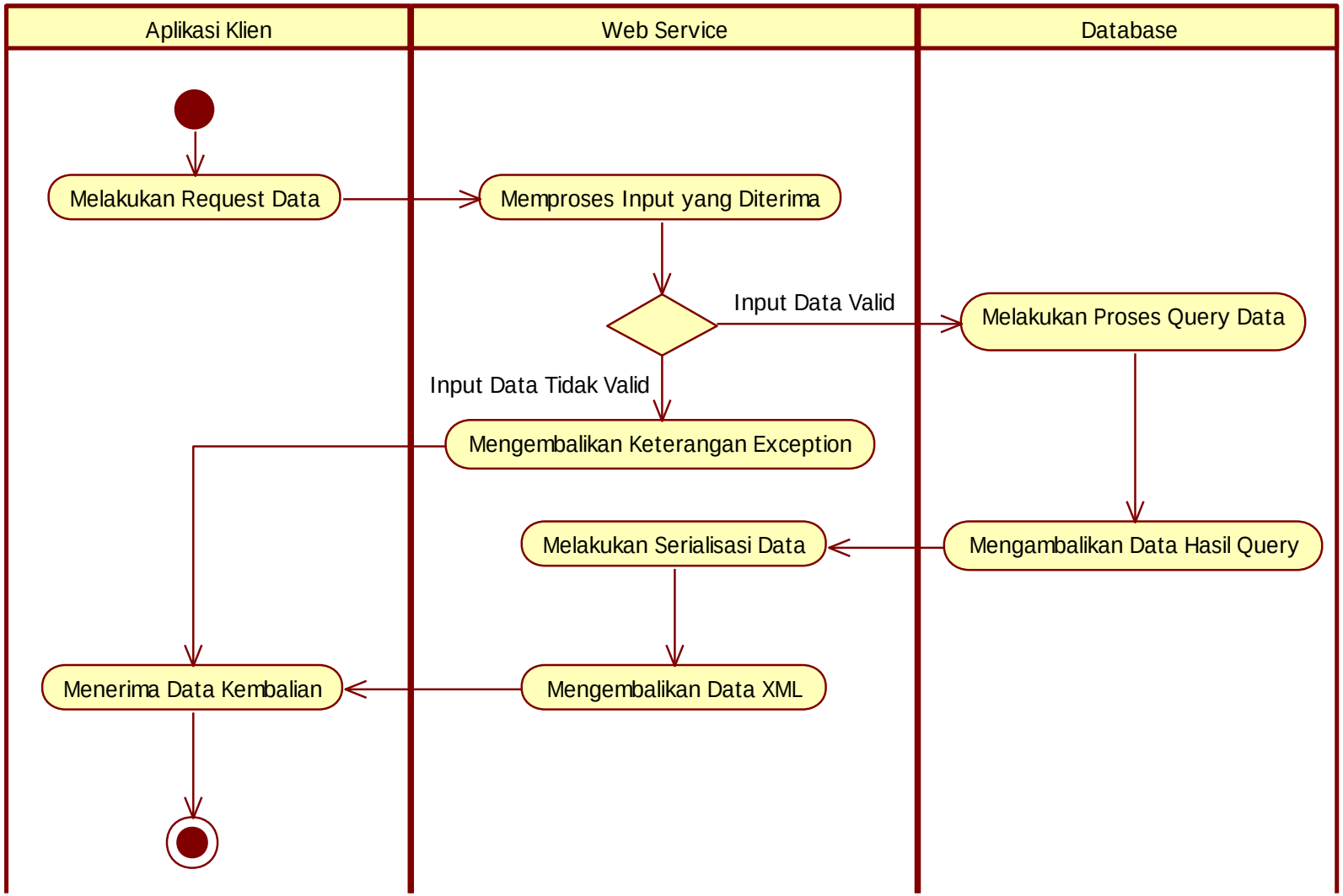
Gambar 9.1. *Block Diagram* Modul *Battle* dan *Shopping*



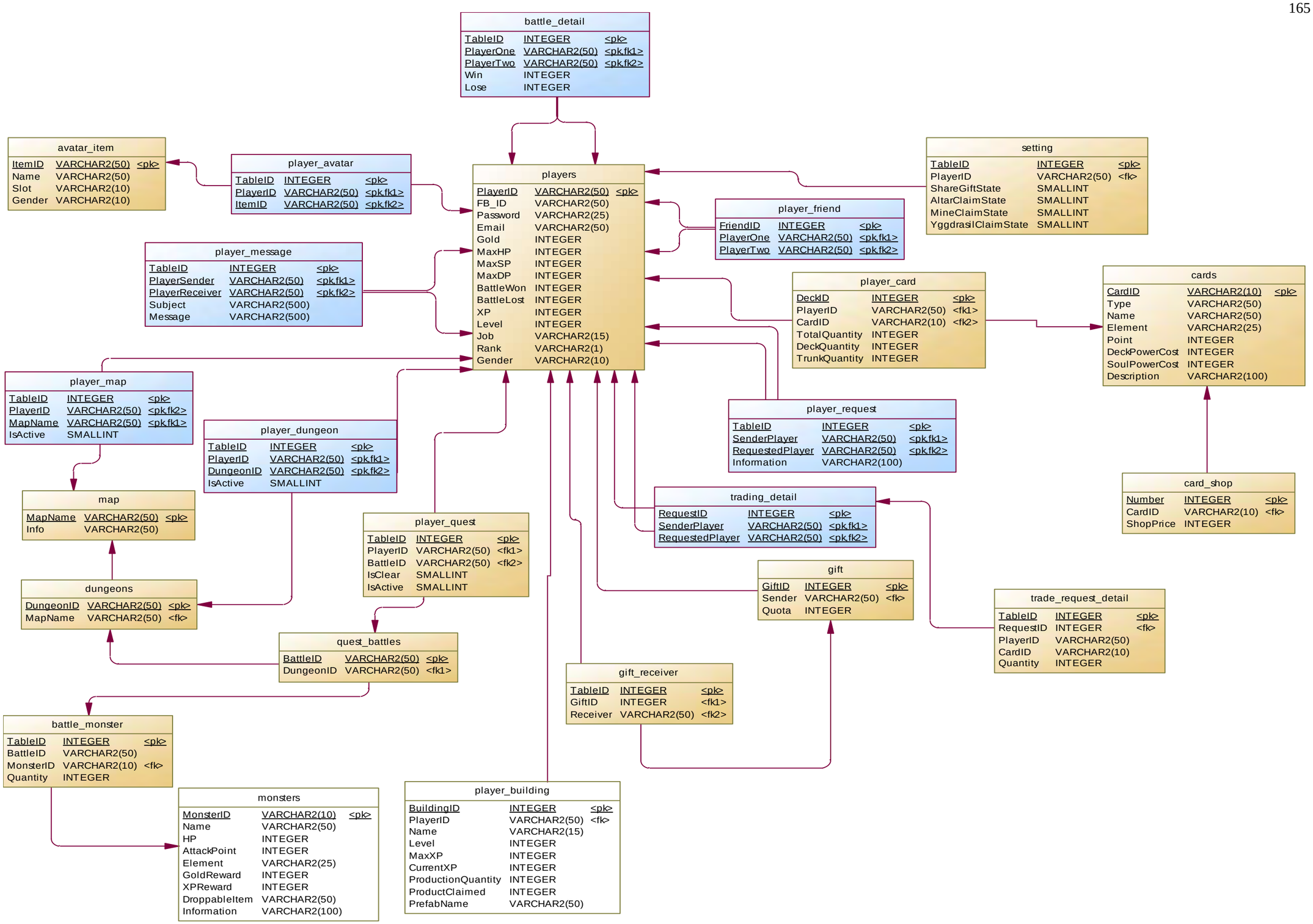
Gambar 9.2. *Block Diagram* Modul *Social*



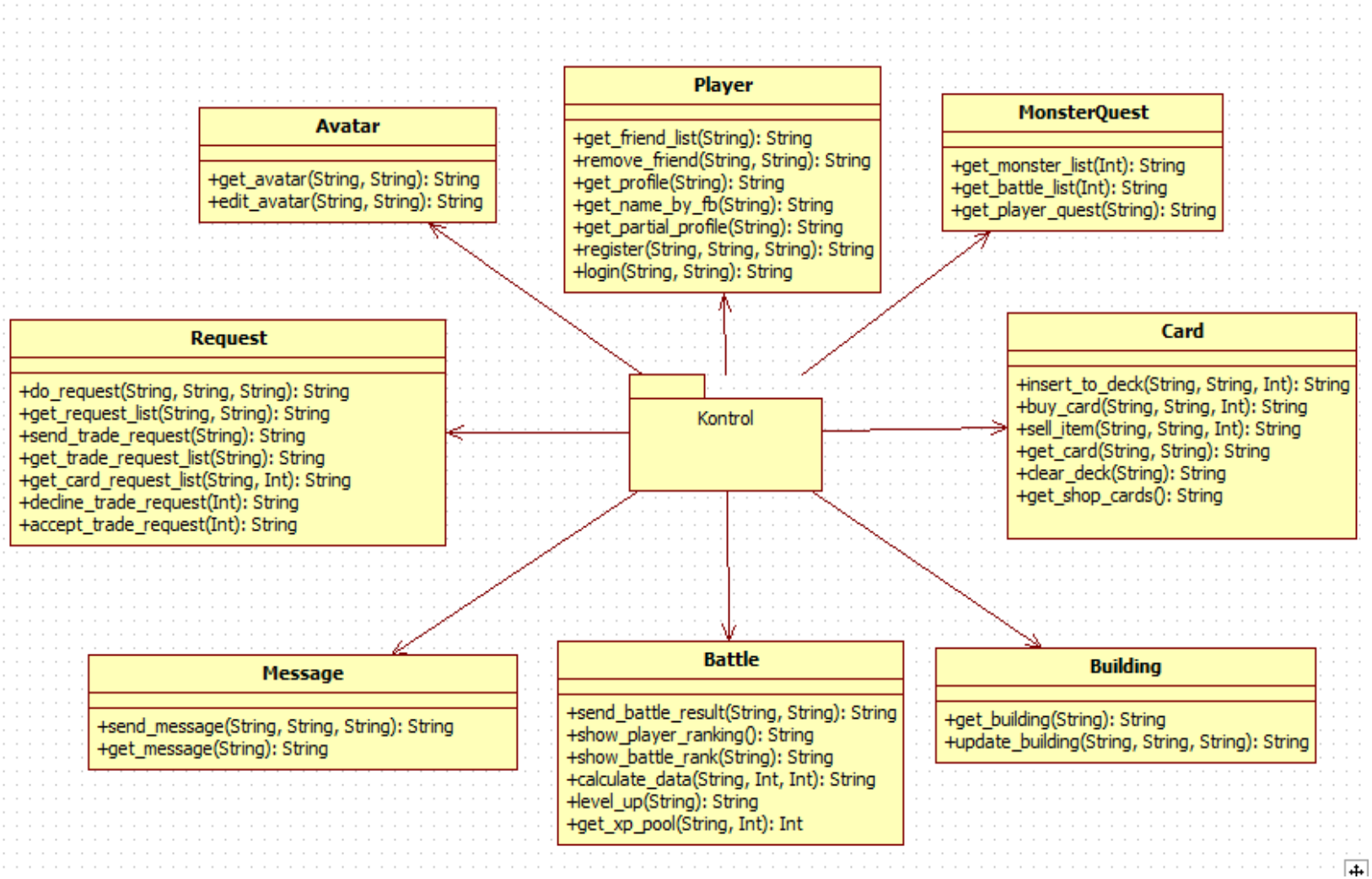
Gambar 9.3. Arsitektur *web service*



Gambar 9.4. Arsitektur Sistem dalam Bentuk *Activity Diagram*



Gambar 9.5. Perancangan Basis Data dalam Bentuk PDM



Gambar 9.6. Diagram Kelas Lapisan Data

DAFTAR PUSTAKA

[1] MySQL Developer Team. What is MySQL. [Online]. http://dev.mysql.com/doc/refman/4.1/en/what-is-mysql.html

[2] MySQL Developer Team. Stored Procedure in MySQL. [Online]. http://dev.mysql.com/doc/refman/5.1/en/create-procedure.html

[3] Pravins. (2013, Aug.) NuSOAP Library for SOAP Server CodeIgniter. [Online]. http://www.php-guru.in/2013/soap-server-in-codeigniter-using-nusoap-library/

[4] R. Veldhuizen. (2011, Jan.) WSDL in NuSOAP PHP. [Online]. http://roelveldhuizen.com/generate-a-wsdl-using-nusoap-in-php/

[5] M. Rouse. SOAP Definition. [Online]. http://searchsoa.techtarget.com/definition/SOAP

[6] L. Revill. (2013, Dec.) CodeIgniter Tutorial. [Online]. http://www.revillweb.com/tutorials/codeigniter-tutorial-learn-codeigniter-in-40-minutes/

[7] N. Lovell. (2011, Jan.) What Is Social Game. [Online]. http://www.gamesbrief.com/2011/01/what-is-a-social-game

[8] Ellislab. Database Call Function Code Igniter. [Online]. https://ellislab.com/codeigniter/user-guide/database/call_function.html

[9] Ellislab. Query Result Code Igniter. [Online]. https://ellislab.com/codeigniter/user-guide/database/results.html

[10] Microsoft. XML Text Reader MSDN Microsoft. [Online]. http://msdn.microsoft.com/en-us/library/system.xml.xmltextreader.aspx

[11] M. Gupta. (2012, Nov.) XML Serialization and Deserialization. [Online]. http://www.codeproject.com/Articles/487571/XML-

Serialization-and-Deserialization-Part-2

[12] J. J. Owen. (2011, Mar.) MySQL Stored Procedure in CodeIgniter. [Online]. http://johnjowens.tumblr.com/post/3625253235/codeigniter-mysql-and-stored-procedures

[13] D. Schroeder. (2014, Mar.) XML as Object in C#. [Online]. http://blog.danskingdom.com/saving-and-loading-a-c-objects-data-to-an-xml-json-or-binary-file/

[14] A. Sofwan. (2007) Belajar Framework CodeIgniter pada PHP. [Online]. http://dppka.jogjaprov.go.id/cmskppd/file/belajar-php-dengan-framework-code-igniter3.pdf

[15] Ellislab. Active Record Code Igniter. [Online]. https://ellislab.com/codeigniter/user-guide/database/active_record.html

[16] A. Villalba. (2013, Jul.) SOAP Server in CodeIgniter. [Online]. http://phpsblog.agustinvillalba.com/creating-a-soap-server-in-codeigniter/

BIODATA PENULIS

Penulis, Muhammad Agil Mahbuby, lahir di kota Bangkalan pada tanggal 22 Juli 1992. Penulis adalah anak kedua dari tiga bersaudara dan dibesarkan di kota Kediri, Jawa Timur.

Penulis menempuh pendidikan formal di SDN Semampir I Kediri (1998-2004), SMPN 3 Kediri (2004-2007), SMAN 7 Kediri (2007-2010). Pada tahun 2010, penulis memulai pendidikan S1 jurusan Teknik Informatika Fakultas Teknologi Informasi di Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur.

Di jurusan Teknik Informatika, penulis mengambil bidang minat Rekayasa Perangkat Lunak dan memiliki ketertarikan di bidang basis data, dan *game*. Penulis dapat dihubungi melalui alamat email agil.mahbuby@gmail.com.